Titre: Title:	Conception d'une architecture multi-agents pour la planification d'événements multi-parties
Auteur: Author:	Younes Zaki
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Zaki, Y. (2003). Conception d'une architecture multi-agents pour la planification d'événements multi-parties [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7003/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7003/
Directeurs de recherche: Advisors:	
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UNE ARCHITECTURE MULTI-AGENTS POUR LA PLANIFICATION D'ÉVÉNEMENTS MULTI-PARTIES

YOUNES ZAKI

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

JANVIER 2003

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

CE MÉMOIRE INTITULÉ :

CONCEPTION D'UNE ARCHITECTURE MULTI-AGENTS POUR LA PLANIFICATION D'ÉVÉNEMENTS MULTI-PARTIES

présenté par : ZAKI Younes
en vue de l'obtention du diplôme de : maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen composé de :

Mme. BOUCHENEB Hanifa, Ph. D., président
M. PIERRE Samuel, Ph. D., membre et directeur de recherche
M. GLITHO Roch, Ph. D., membre et codirecteur de recherche
M. QUINTERO Alejandro, Ph. D., membre

REMERCIEMENTS

J'aimerais exprimer mes remerciements et ma profonde reconnaissance à mon directeur de recherche, le professeur Samuel Pierre et à mon codirecteur M. Roch Glitho pour leur générosité et le temps qu'ils m'ont consacré. Tous deux ont été une source d'inspiration et joué le rôle de conseiller tout le long de l'élaboration de ma maîtrise et ce, avec énormément de patience et un grand intérêt.

Je remercie aussi tous les membres du laboratoire LARIM, pour leur esprit d'équipe et l'atmosphère sympathique qu'ils ont su maintenir dans un environnement de recherche universitaire. Je me rappellerai particulièrement des longues discussions avec mes collègues Ali Chamam, Fabien Houéto, Rolland Mewanou, Edgar Olougouna et Yufei Wu, sans oublier le support technique de Mme Sabine Kébreau.

Je suis aussi très reconnaissant envers ma famille et mes amis, plus particulièrement envers mes parents pour m'avoir appris à persévérer et pour la confiance invariable qu'ils ont portée en moi. Je les remercie du fond du cœur pour l'aide sur le plan moral ainsi que financier qu'ils m'ont accordée. Leur soutien m'a été d'une grande valeur.

RÉSUMÉ

L'utilisation grandissante des ordinateurs personnels et l'intérêt croissant pour la technologie d'agents, tous deux couplés à l'évolution des environnements réseau en terme de taille de données échangées et de nombre d'utilisateurs, vont graduellement aider les agents à remplacer les humains et à accomplir les tâches de ces derniers de façon autonome. La planification d'évènements multi-parties est un exemple d'application répartie où cette automatisation par les agents pourrait avoir lieu. En effet, les agents mobiles conviennent bien aux exigences de la planification répartie, et ce, pour deux raisons. Premièrement, à cause de leur capacité à se déplacer vers les serveurs de calendriers où les agendas personnels sont stockés, permettant ainsi de diminuer la largeur de bande requise et d'assurer une confidentialité aux informations accédées; deuxièmement, en raison de leur robustesse face à des mauvaises liaisons.

Par ailleurs, le contexte actuel de mondialisation des marchés forcera une plus grande interaction entre les compagnies, produisant ainsi des problème de planification plus complexes et plus exigeants en temps. Ce fait joue en faveur des solutions à base d'agents mobiles, les présentant comme une meilleure alternative par rapport à l'approche client/serveur grâce, notamment, à l'automatisation de tâche et à la confidentialité de l'accès que ces premiers permettent.

Dans ce mémoire, nous élaborons des solutions à base d'agents utilisant des architectures mono-agent et multi-agents pour accomplir la tâche de planification. Le but poursuivi est de présenter une évaluation de performances de ces solutions en terme de temps requis pour la planification et de la charge générée par l'application sur le réseau. La conception d'une architecture multi-agents utilisant un algorithme de coordination inter-agent et employant une partie d'une ontologie élaborée pour l'accomplissement de la planification a permis de constater la capacité des SMA à s'adapter à des tâches réparties; elle démontre aussi, grâce au partage centralisé de l'information, la robustesse et le degré d'automatisation qu'offrent de tels systèmes.

Les mesures de temps nécessaire à l'exécution de l'application répartie et les mesures de charge générée sur le réseau par cette dernière ont permis de préciser le coût des échanges entre agents dans un système multi-agents et de déterminer l'utilité du recours à la partition des données traitées en fonction de la taille de ces données et du degré de granularité choisi. Ainsi, nous avons pu constater avec le SMA un gain de performance pour les deux métriques par rapport au mono-agent en réduisant le nombre de communications.

Nos mesures de performance ont permis d'identifier les coûts imputables à plusieurs opérations présentes dans le paradigme agent tels la communication inter-agent et la migration, et ce, en fonction de la nature et la taille des données échangées. Elles offrent aussi une bonne appréciation des multiples possibilités qu'offrent les agents pour la résolution répartie de problèmes.

ABSTRACT

The wide use of personal computers and the growing interest for the software agent technology, coupled to the evolution of network environments in terms of size of data exchanged and number of users, will gradually help agents replace humans and autonomously perform their tasks. Distributed meeting scheduling (DMS) is an example of distributed applications where this automation by software agents could take place. Mobile agents well suit the requirements of DMS. This is due to their capability of moving to the calendar servers where the personal agendas are stored – therefore saving bandwidth and increasing information confidentiality – and also to their robustness in the occurrence of connections loss.

Furthermore, the globalization of markets will force a growing interaction between companies, therefore, making scheduling problems more complex and time consuming. This fact helps to outline the benefits of software agent-based solutions, and makes mobile agents a very attracting alternative to the client/server paradigm due to the automation of task and the confidentiality of access to information that the first offers.

In this work, we propose to present solutions based on mobile agents using mono-agent and multi-agent strategies for DMS problem resolution. The main goal is elaborating a performance evaluation of this solutions in terms of time needed for scheduling and network load generated by the application. Our performance measures helped identifying the costs of many operations used by the agent-based paradigm like inter-agent communications and agent migration in relation to the nature and size of information exchanged, it also gives a general appreciation of the possibilities that software agents offer for resolving distributed problems.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES ET ABRÉVIATIONS

AC	Agent Client
AP	Agent Planificateur
API	Application Programming Interface
IA	Intelligence Artificielle
JDK	Java Development Kit
KIF	Knowledge Interchange Format
KQML	Knowledge Query Manipulation Language
ORB	Object Request Broker
PDA	Personal Data Assistant
PDL	Première Date Libre
RMI	Remote Method Invocation
SSL	Secure Socket Layer
SMA	Système Multi-Agents
TCP	Transmission Control Protocol
XML	eXtended Markup Language

CHAPITRE I

INTRODUCTION

L'évolution rapide de la technologie de réseau, couplée à la croissance exponentielle des services et de l'information disponibles sur l'Internet, annonce une nouvelle ère d'une informatique omniprésente. Des centaines de millions d'usagers auront dans l'avenir accès à une quantité impressionnante d'information par l'intermédiaire d'une variété de terminaux. Ces terminaux ne seront plus de puissants ordinateurs fixes et isolés, mais plutôt des dispositifs portables qui vont accompagner l'usager (ordinateurs de poche, assistants personnels, nouvelle génération de téléphones, etc.); ils bénéficieront d'une faible bande passante, d'une faible capacité de calcul et impliqueront un constant changement de la taille et de la topologie du réseau. Cette situation défie les techniques traditionnelles de conception de systèmes répartis et impose de nouveaux paradigmes qui pourront s'adapter aux changements des conditions du service. La coopération dans les systèmes multi-agents mobiles constitue un de ces nouveaux paradigmes et fera l'objet de ce mémoire. Ce chapitre définit d'abord les notions de base permettant de préciser la problématique de recherche, décrit ensuite les objectifs visés, pour finalement esquisser un aperçu du plan de ce mémoire.

1.1 Définitions et concepts de base

Les applications réparties traditionnelles sont conçues en tant qu'un groupe de processus assignés statiquement à un environnement d'exécution donné et qui coopèrent en ignorant l'état du réseau. Par contre, le paradigme *agent mobile* introduit des applications composées d'entités conscientes du réseau (*agents*), capables de changer d'environnement d'exécution par le transfert de leur code exécutable à un site différent (*mobilité*). Cette conscience de l'état du réseau, associée à la mobilité, distingue les

systèmes d'agents mobiles des autres systèmes répartis et environnements de programmation.

Les systèmes d'agents mobiles modélisent le réseau en une connexion de plates-formes d'exécution. La migration entre ces plates-formes est activée par les agents eux-mêmes à travers l'invocation d'une commande à l'intérieur de leur code. La mobilité de l'agent implique la migration non seulement du code (comme cela est le cas des *applets* java), mais aussi de l'état interne de l'agent ou l'état des données que ce dernier transporte.

Le passage de l'architecture client-serveur, le modèle le plus utilisé en applications réparties, au paradigme d'*agent mobile* peut être justifié par les multiples avantages que présente ce dernier. Les plus évidents de ces avantages sont la possibilité d'opérer malgré la faible qualité de la connexion en terme de fiabilité et de bande passante, ainsi que la réduction de la charge de calcul pour le terminal usager. Ces avantages se prêtent au domaine des usagers mobiles plus qu'à tout autre domaine. En effet, les usagers se déplaçant sur le réseau subissent les limitations des liaisons généralement radio à faible bande passante et utilisent des machines portables à faible capacité de traitement.

Toutefois, les agents mobiles peuvent s'avérer plus efficaces que les modèles traditionnels dans plusieurs autres domaines. En effet, leur grande flexibilité leur permet de s'adapter au profil de l'usager et ainsi devenir très utiles dans des applications de recherche et de filtrage d'information, de gestion d'agenda électronique, ainsi que dans plusieurs autres applications permettant de personnaliser le comportement du serveur vis-à-vis du client.

1.2 Éléments de la problématique

La dernière décennie a été porteuse de plusieurs débats dans le domaine de l'informatique mobile. La majorité de ces débats, arguant des mérites du paradigme agent mobile, promettaient une amélioration des applications dans les environnements répartis. Toutefois, il n'existe jusqu'à nos jours aucune fonction client-serveur qui soit pratique aux applications réseaux et implémentée uniquement par l'usage d'agents

mobiles. Pour toute application basée sur les agents mobiles, il existe une alternative basée sur des protocoles déjà développés. Plus encore, le manque d'expérimentation sur ces applications pouvant prouver les avantages du paradigme agent mobile constitue un obstacle à son éventuelle succession à l'approche client-serveur.

Ceci dit, le potentiel des agents mobiles n'en est pas moins réduit. L'utilisation de plus en plus grande de dispositifs portables et l'expansion importante des réseaux mobiles, toutes deux couplées à l'apparition de plusieurs plates-formes d'agents dont quelques-unes sont déjà commercialisées (ex : enago de IKV++), promettent l'apparition ainsi que l'adoption d'applications mobiles basées sur les agents. Pour être convaincu de ce point, il ne faut pas oublier les motivations qui sous-tendent l'apparition des agents mobiles, soit la réduction de la charge de la communication et le transfert du calcul vers le serveur.

Ce mémoire se veut donc une contribution au domaine des applications mobiles basées sur les agents, en essayant de les adapter aux exigences des équipements mobiles et de tirer profit du profil de ces derniers. Il s'inscrit dans le prolongement de plusieurs travaux dont principalement celui de Olougouna (2001) qui a exploité le fait que ces moniteurs d'information intègrent des calendriers personnels pour développer une solution d'agents mobiles permettant, entre autres, la planification de rendez-vous.

La solution présentée par Olougouna consistait à envoyer un agent mobile sur le réseau. Ce dernier parcourait les serveurs et accédait aux agendas localement. Ainsi, il pouvait contourner les limitations des architectures client-serveur qui s'appuyaient sur un partage centralisé de l'information et un fonctionnement manuel. Cependant, les résultats obtenus, quoique prometteurs, ne semblent pas trancher de façon décisive pour le paradigme agent mobile. Une manière de mettre l'accent sur le gain en temps et en charge consiste à envoyer plusieurs agents qui se partagent la tâche sur le réseau en se répartissant les serveurs et en échangeant les informations.

1.3 Objectifs de recherche

Ce mémoire se donne comme objectif de concevoir une architecture répartie basée sur les agents mobiles et qui a pour but de se servir des avantages que peut apporter la collaboration de plusieurs agents à une application bien définie qui est la planification électronique d'événements. De manière plus spécifique, cette recherche vise à :

- concevoir une version multi-agents de l'implémentation mono-agent du planificateur électronique d'événements ;
- étudier les performances de l'approche multi-agents mobile pour l'application voulue par rapport d'abord au prototype mono-agent, ensuite à la méthode client-serveur, et ce, pour la même application ;
- évaluer les avantages et les inconvénients de la coordination inter-agent dans un système multi-agents en termes de charge de communication et de temps d'exécution.

L'architecture visée doit être évolutive et permettre l'évaluation des points suivants : (i) l'avantage d'un système multi-agents sur celui mono-agent, (ii) la performance de l'architecture multi-agents par rapport à l'approche client-serveur.

1.4 Plan du mémoire

Ce mémoire se présente en cinq chapitres. Le Chapitre II se propose de cerner plus en détail les notions entourant le paradigme d'agent mobile; il introduit aussi le concept de système multi-agents, pour ensuite faire le tour de ce qui a été réalisé dans le domaine des agents. Le Chapitre III décrit pour commencer l'application de planification électronique d'évènements, présente ensuite le modèle mono-agent existant, pour enfin exposer l'architecture développée ainsi que le modèle de coordination et de communication retenu pour cette dernière. Comme suite logique au chapitre qui le précède, le Chapitre IV donne l'évaluation ainsi que les mesures expérimentales de l'approche multi-agents, tout en les confrontant aux résultats des deux autres alternatives, la mono-agent et l'approche client-serveur. Finalement, le Chapitre V, en

guise de conclusion, présente une synthèse des travaux assortie d'une discussion sur le prix à payer pour la coordination dans un système multi-agents, et donne des indications pour des travaux futurs.

CHAPITRE II

SYSTÈMES MULTI-AGENTS

Depuis quelques années, une orientation nouvelle de la programmation se précise, dite *Programmation Orientée Agent* (POA). Elle émerge d'une conception comportementaliste des objets informatiques par opposition à la *Programmation Orientée Objet* (POO) qui produit des applications desquelles ne ressortent que des comportements attendus. Cette orientation répond au concept d'*agent autonome* et doté de but propre par opposition à l'objet. Les comportements collectifs produits par les interactions des *agents* ont crée une nouvelle discipline, celle des *systèmes multi-agents* (SMA). Ce chapitre introduit, tout d'abord, les notions d'*agents* et de *systèmes multi-agents*, et fournit ensuite une évaluation des avantages et limitations de ces nouveaux paradigmes. Par la suite, il donne un aperçu des applications basées sur les *agents* et sur les *SMA*, et à la fin, il offre une synthèse des problèmes ouverts.

2.1 Agents

Dans cette section, nous offrons une définition du paradigme *agent*, par la suite nous établirons certaines caractéristiques de ce dernier ainsi que les avantages et les inconvénients qu'il présente.

2.1.1 Concept d'agent

Un agent peut être défini comme une entité (physique ou abstraite) capable d'agir sur elle-même et son environnement, disposant d'une représentation partielle de cet environnement, pouvant communiquer avec d'autres agents et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents (Ferber et al., 1988). Un agent peut aussi être défini comme une aide logicielle qui remplace l'utilisateur dans une tâche routinière et pénible, comme

organiser l'horaire de réunions ou le tri du courrier (électronique), ou qui cherche et trie des informations correspondant aux intérêts de l'utilisateur.

Les agents ont deux tendances : une tendance sociale tournée vers la collectivité (les mécanismes et connaissances associés concernent les activités du groupe) et une tendance individuelle avec des mécanismes et des connaissances contenant les règles de fonctionnement interne de l'agent. On peut caractériser un agent par : son rôle, sa spécialité, ses objectifs et ses fonctionnalités, ses croyances, ses capacités décisionnelles, ses capacités de communication et éventuellement ses capacités d'apprentissage.

2.1.2 Caractéristiques d'un agent

Définissons d'abord les caractéristiques minima d'un agent, qui sont :

L'autonomie : L'agent travaille sans intervention directe, jusqu'à un point défini par l'utilisateur. L'autonomie d'un agent peut aller du simple lancement d'une sauvegarde la nuit, à la négociation du prix d'un produit choisi par son mandataire.

L'interactivité : L'agent doit pouvoir exercer des actions sur son environnement et réciproquement l'environnement peut agir sur l'agent. L'interactivité d'un agent de sauvegarde, par exemple, sera sa possibilité de sauvegarder un certain nombre de fichiers par une action exercée au travers du système d'exploitation. Ce dernier pourra informer l'agent d'une permission non accordée (exemple : un fichier dont l'utilisateur n'a pas de droit de lecture).

La réactivité : L'agent percevra son environnement (qui pourra être l'utilisateur au travers d'une interface graphique, un ensemble d'agents, ...) en répondant dans les temps opportuns aux changements qui surviennent sur cet environnement (exemple : Un agent de sauvegarde pourra exécuter sa tâche à une heure donnée).

Les caractéristiques suivantes peuvent être ajoutées aux caractéristiques de base pour permettre à l'agent d'exécuter sa tâche. Il s'agit ici d'une liste non exhaustive, d'autres caractéristiques peuvent s'ajouter et se combiner.

Coordination : L'agent est capable de coordonner ses actions par rapport à un utilisateur ou un autre agent.

Compétition : L'agent est capable d'agir dans un environnement où d'autres agents interviennent. Le but est le même pour tous les agents présents, mais un seul l'atteindra. Les autres échoueront et, forcément, tous les coups sont permis.

Mobilité : Afin d'éclaircir la définition de mobilité et dissiper nombre de confusions, nous devons prendre en compte deux types de mobilité : la mobilité relative ou par requêtes, la mobilité réelle de l'agent.

- Mobilité relative ou par requêtes : Dans ce cas il n'y a pas un réel déplacement de l'agent. Celui-ci lance une succession de requêtes à destination de différents serveurs. C'est le cas des agents de recherche, tel que Copernic, qui interroge différents moteurs de recherche afin de fournir à son utilisateur une synthèse des résultats.

- Mobilité réelle de l'Agent : Le processus agent se déplace d'un serveur à un autre sur le réseau. Le code de l'objet est transporté ainsi que ses données. L'exécution de l'agent se poursuit sur la nouvelle machine.

Tout le long de ce mémoire, c'est ce dernier cas qui sera pris en compte quand il s'agira d'agent mobile.

2.1.3 Agents cognitifs vs réactifs

La complexité des fonctionnalités des agents impliqués dans une application varie selon deux écoles coexistante aujourd'hui : l'école cognitive et l'école réactive. Suivant le type d'agent utilisé, on parlera de systèmes cognitifs ou de systèmes réactifs.

Agents cognitifs

Les systèmes d'agents cognitifs sont fondés sur la coopération d'agents capables a eux seuls d'effectuer des opérations complexes. Un système cognitif comprend un petit nombre d'agents qui disposent premièrement d'une capacité de raisonnement sur une base de connaissances, deuxièmement d'une aptitude à traiter des informations diverses liées au domaine d'application et, finalement, d'informations relatives à la gestion des interactions avec les autres agents et l'environnement. Chaque agent est assimilable suivant le niveau de ses capacités a un système expert plus ou moins sophistiqué, on parle d'agent de forte granularité (coarse grain).

Agents réactifs

Les défenseurs de cette approche partent du principe suivant : dans un système multi-agents, il n'est pas nécessaire que chaque agent soit individuellement intelligent pour parvenir à un comportement global intelligent. En effet, des mécanismes simples de réactions aux événements peuvent faire émerger des comportements correspondant aux objectifs poursuivis. Cette approche propose la coopération d'agents de faible granularité (fine grain) mais en nombre beaucoup plus grands que celui de l'approche cognitive.

Les agents réactifs sont de plus bas niveau, ils ne disposent que d'un protocole et d'un langage de communication réduits, leurs capacités répondent uniquement à la loi stimulus/action.

2.1.4 Avantages et inconvénients

Avant de discuter des avantages et inconvénients des agents mobiles, il faut d'abord mentionner le modèle client/serveur, qui est certainement le modèle le plus utilisé pour la construction d'applications réparties. Dans cette architecture, le serveur procure un ensemble fixe d'opérations, toutes les autres doivent être exécutées par le client. Le client émet une requête vers le serveur grâce à son adresse et le port qui désigne un service particulier du serveur. Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client et son port. Les échanges requêtes/réponses dureront tant que l'opération procurée par le serveur ne correspond pas aux besoins du client.

Après cette brève précision, nous pouvons énumérer les avantages ainsi que les inconvénients que présentent les agents mobiles.

D'abord citons les avantages :

- Les agents mobiles permettent de déplacer l'exécution vers les serveurs et de diminuer ainsi le coût d'accès à ces derniers.
- Les agents mobiles permettent une interaction asynchrone, ils implantent la programmation à distance au lieu des appels de fonction à distance. Ceci est particulièrement utile pour les palmtops et ordinateurs mobiles qui ne sont connectés que de façon intermittente.
- Les agents rendent les systèmes plus flexibles en permettant une personnalisation des interactions (Harisson et al., 1995).
- Les agents mobiles permettent d'encapsuler l'information et de la transporter vers une destination voulue, ils réduisent ainsi la charge du réseau causée par les protocoles de communication sur lesquels reposent les interactions des systèmes répartis.
- Les agents mobiles permettent de réduire le flot de données brutes sur le réseau en manipulant localement les données de grande taille.

- Les agents mobiles permettent de dépasser les limitations de l'ordinateur client en exécutant les tâches sur un serveur à plus grande capacité de calcul, de mémoire et de communication.
- Les agents mobiles permettent une indépendance par rapport au système d'exploitation.

Ceci dit, les agents mobiles ont tout de même quelques lacunes, parmi lesquelles :

- La taille du code : la taille de l'agent peut parfois dépasser celle des données accédées, ce fait affecte le gain de bande passante.
- Manque d'une *killer application* : Il n'existe pas encore d'application populaire démontrant la supériorité du modèle basé sur les agents mobiles (Kotz et al, 1999).
- La sécurité : cette lacune est présentée comme étant la lacune majeure dans les systèmes d'agents mobiles. Une contrainte de sécurité qui revient le plus souvent est de protéger l'agent d'un hôte ou d'un autre agent mal-intentionné.
- Manque de plate-forme standard : ils existent actuellement plusieurs plate-formes qui permettent de créer des agents mobiles, ces derniers ne pourront s'exécuter que sur des hôtes ayant cette même plate-forme qui les a créés.

2.2 Systèmes multi-agents

Contrairement aux systèmes d'Intelligence Artificielle qui simulent jusqu'à un certain point le raisonnement humain, les SMA sont conçus comme un ensemble d'agents interagissant selon des modes de coopération, de concurrence ou de coexistence. Cette section donne une définition d'un SMA, précise l'intérêt derrière d'un tel système et les défis qu'il doit relever et, pour finir, présente ses principales caractéristiques.

2.2.1 Définition

Un système multi-agents est un système informatique qui se distingue par son environnement logiciel et matériel, par ses éléments de base (ses agents) et par les

mécanismes d'actions et d'interactions existant entre ces éléments et l'environnement en question. L'environnement d'un système multi-agents est défini par une description détaillée de toutes les ressources matérielles (mémoire, ordinateurs, réseaux, etc.) et logicielles (bases de données, systèmes d'exploitation, bases de connaissances, etc.) mises à la disposition du système. Comme le sous-entend son nom, les éléments de base d'un système multi-agents sont les agents. Ces derniers sont les entités logicielles qui caractérisent, par leurs actions sur l'environnement et leurs interactions, l'activité principale du système. Finalement, les mécanismes d'actions des agents sur l'environnement et les mécanismes d'interactions entre les agents sont souvent régis par des règles préétablies du type événement-action et d'un langage de communication (KQML, KIF, etc.)

2.2.2 Intérêt des SMA

L'émergence de systèmes informatiques de plus en plus complexes et souvent répartis sur plusieurs sites a conduit à l'apparition d'applications requérant une activité complexe et coordonnée. Pour la résolution de telles tâches, il est avantageux de faire coopérer des ensembles d'agents dotés d'un minimum d'intelligence et de coordonner leurs buts. Pour ce faire, la solution la plus appropriée est le recours à un système multi-agents.

Les SMA sont des systèmes idéaux pour représenter des problèmes possédant de multiples méthodes de résolution, de multiples perspectives et/ou de multiples résolveurs. Ces systèmes possèdent les avantages traditionnels de la résolution répartie et concurrente de problème comme la modularité, la vitesse (grâce au parallélisme), et la fiabilité (due à la redondance). Ils héritent aussi des bénéfices de l'Intelligence Artificielle, comme la facilité de maintenance, la réutilisation et la portabilité mais surtout, ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués. Les types courants d'interactions incluent la coopération, la coordination et la négociation.

2.2.3 Défis des SMA

Bien que les SMA offrent de nombreux avantages, ils doivent aussi relever beaucoup de défis. Les principaux problèmes inhérents à la conception et à l'implémentation des SMA sont les suivants (Bond et al.,1988 ; Franklin et al., 1997 ; Iglessias et al., 1997) :

- La manière de formuler, décrire, décomposer, et allouer les problèmes et synthétiser les résultats.
- La façon de permettre aux agents de communiquer et d'interagir, savoir quoi communiquer et quand.
- La garantie de la cohérence entre les actions des agents.
- La manière de permettre aux agents individuels de représenter et de raisonner sur les actions, plans et connaissances des autres agents afin de se coordonner avec eux.
- La reconnaissance des intentions conflictuelles dans un ensemble d'agents essayant de coordonner leurs actions.
- Trouver le meilleur compromis entre le traitement local au niveau d'un seul agent et le traitement réparti entre plusieurs agents.

2.2.4 Caractéristiques des SMA

Les caractéristiques principales des SMA sont les suivantes :

- Chaque agent ne dispose que de parties d'informations incomplètes et possède un champ d'action limité ;
- Le contrôle du système est réparti ;
- Les données manipulées sont décentralisées ;
- Les traitements sont asynchrones.

Les SMA peuvent se distinguer les uns des autres au niveau des agents eux-mêmes, des interactions entre les agents et des environnements dans lesquels évoluent

les agents. Le Tableau 2.1 donne une idée des différents types de SMA en fonction des différentes valeurs de ces caractéristiques (Weiss 1999).

Tableau 2.1 Les types de SMA

	Attribut	Valeurs
Agents	- Nombre	Deux et plus…
	- Uniformité	Homogénéité … hétérogénéité
	- Buts	Contradictoires … complémentaires
	- Architecture	Réactive … délibérative
	- Compétences	Simple … avancée
Interaction	- Fréquence	Faible … élevée
	- Persistance	Court-terme … long-terme
	- Niveau	Envoi de signal … manipulation des connaissances.
	- Contrôle et flux de données	Décentralisé … hiérarchique
	- Variabilité	Fixe … variable
	- Propos	Compétitif … coopératif
Environnement	- Prédiction	Possible … impossible
	- Accessibilité et connaissances	Illimitées … limitées
	- Dynamique	Fixe … variable
	- Variété	Pauvre … riche

2.3 Coordination inter-agents

Quelle que soit la définition donnée d'un système multi-agents, on retrouve toujours une mention du besoin de coordination et/ou coopération entre les composants de ce système, c'est-à-dire les agents. Cette section est dédié donc à définir les multiples concepts nécessaires pour établir une coordination entre les agents dans un environnement multi-agents.

2.3.1 Définition de la coordination

La coordination est la propriété d'un système d'agents accomplissant une activité en environnement partagé (Huhns et al., 1999). L'objectif est d'avoir un système plus cohérent. Le degré de coordination peut être envisagé comme la quantité d'activités supplémentaires nécessaire pour éviter les activités redondantes.

Les actions des agents doivent être coordonnées pour plusieurs raisons :

- Il y a des dépendances entre les actions des agents.
- Aucun agent n'a suffisamment de compétence, de ressources et d'information pour atteindre tout seul le but du système complet.
- Il faut éviter les redondances dans la résolution des problèmes.

Dans les systèmes d'intelligence artificielle distribuée, la coordination des actions des agents peut s'organiser suivant deux schémas principaux (Ferber et al., 1990): une coordination au moyen d'un système capable de déterminer et de planifier (globalement) les actions des différents agents, ou à l'inverse, on décide de donner une totale autonomie aux agents qui, à leur tour, identifient les conflits pour les résoudre localement.

On peut distinguer deux types de coordination : la coordination due à la gêne (problème de navigation : les agents doivent coordonner leurs plans de navigation pour s'éviter mutuellement) et la coordination due à l'aide (la manutention : dans un environnement multi-robots, les agents doivent synchroniser leurs actions pour pouvoir agir efficacement et transporter un objet).

2.3.2 Modèle de coordination

Ainsi qu'on l'a déjà mentionné, la coordination est l'art de gérer des interactions parmi des activités, c.-à-d., dans le contexte des systèmes multi-agents, parmi des agents. Un modèle de coordination fournit un cadre formel dans lequel l'interaction des agents peut être exprimée. De façon générale, un modèle de coordination traite de la

création et la destruction des agents, de leurs activités de communication, de leur distribution, ainsi que la synchronisation et la distribution de leurs actions dans le temps.

De manière plus précise, un modèle de coordination se compose de trois éléments (Ciancarini et al., 1999), le modèle est illustré à la Figure 2.1 :

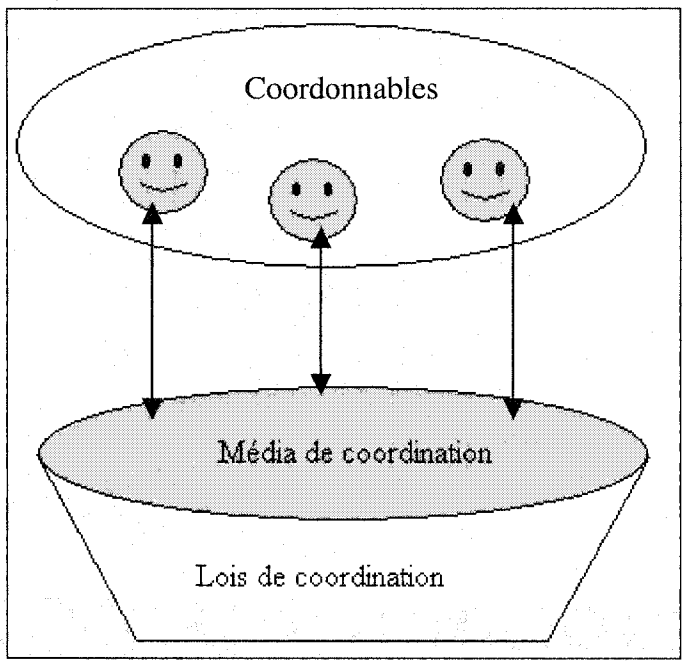

Figure 2.1 Modèle de coordination

- Les objets coordonnables : ce sont les entités dont l'interaction est gérée par le modèle. Ceux-ci pourraient être des processus, des objets, même des utilisateurs et, pour notre cas, des agents ;
- Le média de coordination : c'est le noyau autour duquel les composants d'un système coordonné sont organisés. Des exemples de médias classiques sont les sémaphores, les moniteurs, ou les canaux ; parmi les médias plus complexes, on peut mentionner les tableaux noirs.
- Les lois de coordination : elles définissent le comportement du media de coordination en réponse aux événements d'interaction. Les lois peuvent être définies en termes de langage de communication, qui est une syntaxe employée

pour exprimer et permuter des structures de données, et un langage de coordination, qui est un ensemble de primitifs d'interaction et de leur sémantique.

2.3.3 Communication

La communication est la base de la résolution coordonnée des problèmes. Elle permet de synchroniser les actions des agents et résoudre les conflits de ressources et de buts par la négociation. Dans les systèmes multi-agents, les agents ne disposent d'aucune mémoire commune. La communication entre les agents repose explicitement sur des mécanismes d'envoi de messages de réception et de synchronisation.

Protocoles de communication

Les protocoles de communication sont les règles nécessaires à la communication ; ils permettent de structurer et d'uniformiser les interactions inter-agents. Un protocole de communication peut être basé sur une architecture standard de communication. Les protocoles de communication inter-agent les plus connus sont KQML et KIF.

KQML (Knowledge Query and Manipulation Language)

KQML (Finin et al., 1994) est un protocole pour échanger de l'information entre agents. Son principal atout est que tout ce qui est nécessaire à la compréhension du message est inclus dans le message lui-même. La forme de base du protocole est représentée à la Figure 2.2.

```
(KQML-performative
        :sender <word>
        :receiver <word>
        :language <word>
        :ontology <word>
        :content <expression> ...)
```

Figure 2.2 Message KQML de base

Les performatifs KQML sont modélisés selon les performatifs des actes de langages. Leur sémantique est indépendante du domaine, alors que celle du message l'est et est définie à l'aide des champs :**content** (le message), :**language** (langage d'expression du message), :**ontology** (le vocabulaire du domaine). Les autres arguments d'un message sont **:sender**, **:receiver**, **: reply-with**, **:in-reply-to**.

KQML peut être utilisé comme langage dans un message KQML : le contenu est donc écrit avec la syntaxe KQML. Émetteur et récepteur doivent se comprendre, l'ontologie doit donc être définie et disponible pour chacun des agents participant dans la communication.

Knowledge Interchange Format (KIF)

KIF est un langage logique pour la description dans le cadre de systèmes experts, bases de données, etc. Il a été conçu comme langage intermédiaire, lisible par un programme et par un humain. KIF est une version préfixée du calcul des prédicats du 1er ordre. La description du langage inclut une spécification pour la syntaxe et une pour la sémantique.

Exemple : (> (* (width chip1) (length chip1)) (*(width chip2) (length chip2))) ou encore (=> (and (real-number ?x) (even-number ?n)) (> (expt ?x ?n) 0)))

Comme on le voit, la syntaxe permet la manipulation de variables, elle inclut l'utilisation de la virgule et d'opérateurs.

Architectures de communication

La communication entre les agents peut être organisée suivant quatre schémas différents (Labidi et al., 1993), les trois premiers schémas à la différence du dernier, contiennent une unité centrale appelée *superviseur*, cette unité est responsable de la gestion de la communication entre les agents. Les paragraphes suivants donnent une description de ces différents modèles ainsi qu'une comparaison entre ces derniers.

Communication en anneau

Le principal avantage de ce modèle de communication est la simplicité du mécanisme d'envoi de messages, il permet aussi de détecter la perte d'un message au cas où le superviseur ne collecterait pas le message qu'il a envoyé après un certain moment. En matière de sécurité des échanges, ce modèle permet de garantir une certaine confidentialité, chaque agent envoie un message à un agent ou reçoit un message d'un agent dont il connaît l'identité. Ainsi, il y a peu de chance qu'un agent qui n'appartient pas au système n'intercepte un message ou qu'il n'en génère un mauvais. Le principe de la communication en anneau est le suivant :

- Le superviseur émet un message vers le premier agent (Agent1).
- Chaque agent (Agent *i*) reçoit le message de son prédécesseur (Agent *i-1*) et le transmet à son successeur (Agent *i+1*).
- Le superviseur collecte le message du dernier agent (Agent *n*).

La Figure 2.3 illustre la manière dont la communication s'établit dans ce cas.

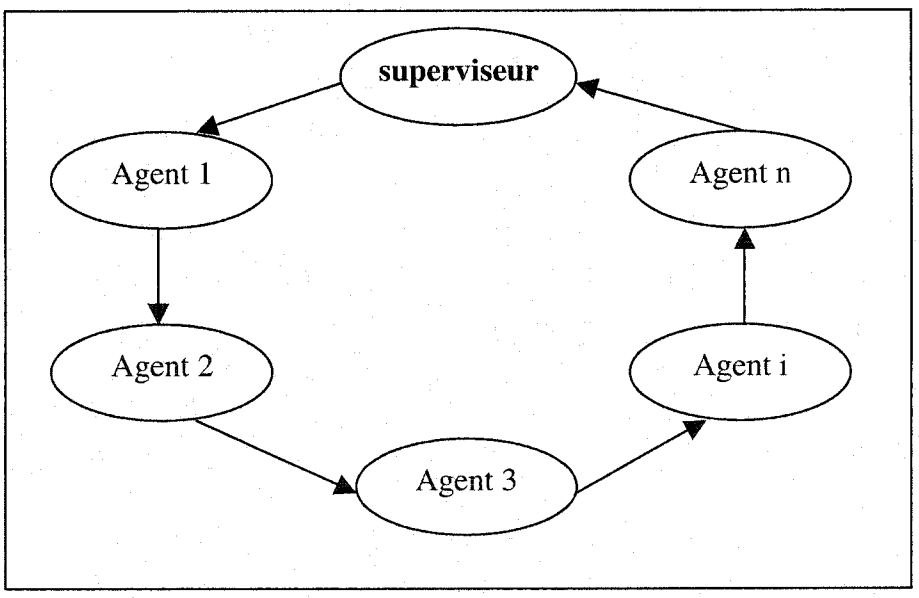

Figure 2.3 Architecture en anneau

Cette manière de communiquer contient plusieurs lacunes. D'abord, les interactions dans ce genre d'organisation sont trop lentes, un message destiné à un agent doit transiter par n agents. Ensuite, dans la mesure où le nombre d'agents est élevé, les agents risquent de saturer le réseau avec des messages inutiles étant donné que plusieurs des agents reçoivent des messages qui ne leur sont pas destinés. Et finalement, le maintien de la structure en anneau devient rapidement un problème complexe quand on prend en compte la mobilité des n agents du système, sachant que chaque agent du système doit connaître l'identité et l'emplacement d'au moins deux autres agents (son prédécesseur et son successeur).

Communication en étoile

Cette méthode de communiquer appelée aussi *Blackboard*, présente l'avantage de l'accès rapide entre le superviseur et les autres agents. Un grand avantage de ce modèle est la simplicité de la structure de communication. En effet, tous les agents du système communiquent de façon bidirectionnelle avec une seule entité, c'est-à-dire le superviseur ; ils n'ont donc pas besoin de connaître l'identité des autres agents, ce qui facilite le maintien de la communication malgré la mobilité des différents agents. La Figure 2.4 illustre ce modèle de communication.

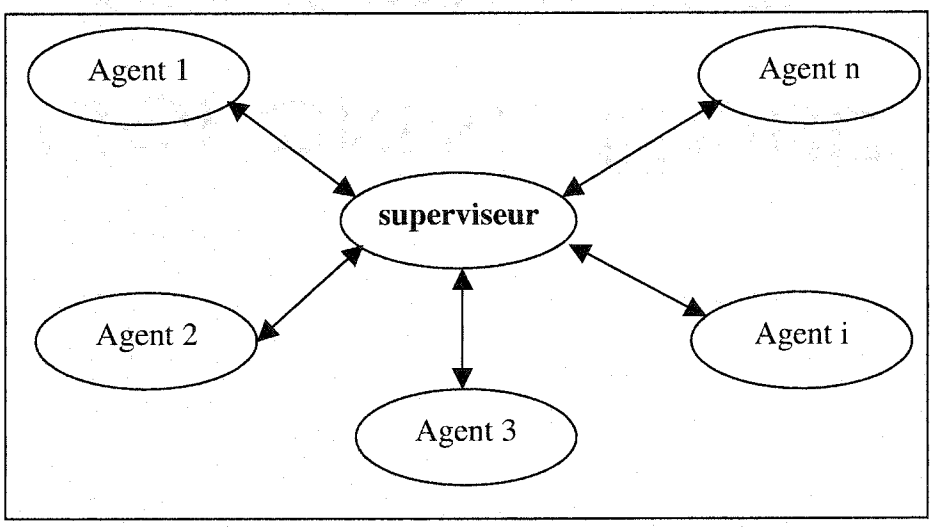

Figure 2.4 Architecture en étoile

Un autre avantage de cette architecture est le faible coût d'envoi et de stockage des messages, car les messages ne sont gardés qu'à un seul endroit, en l'occurrence l'emplacement du superviseur.

Par contre, la nature bidirectionnelle des connexions rend complexe la gestion des interactions inter-agents, le superviseur doit implémenter un mécanisme avancé pour organiser les différents échanges. De plus, la nature des messages étant anonyme, le superviseur doit aussi assurer la sécurité des échanges, en empêchant les agents n'appartenant pas au groupe de s'introduire dans les interactions.

Communication hybrides

La Figure 2.5 illustre ce type d'organisation. C'est une solution hybride reliant deux types d'organisation : un réseau en bus et un réseau en étoile. Cette architecture se retrouve à mi-chemin entre les deux précédentes, elle permet ainsi de présenter un compromis entre leurs inconvénients. Elle hérite de la simplicité de l'architecture en étoile, mais le fait que le superviseur émet un message à tous les agents à chaque échange rend le coût de la communication très élevé pour un nombre important d'agents.

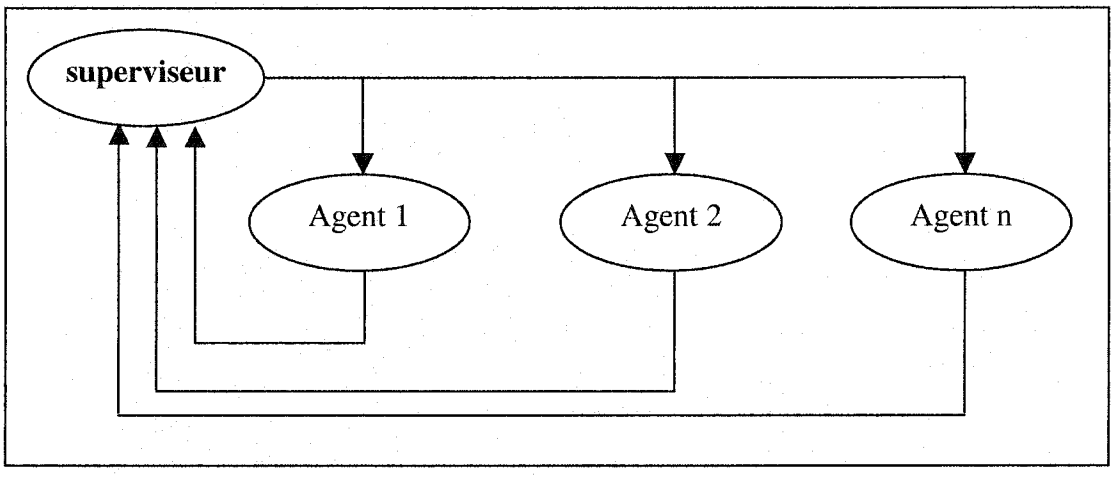

Figure 2.5 Architecture hybride

Communication décentralisée

Ainsi qu'il a été mentionné, ce modèle de communication ne possède pas d'unité centrale ou de superviseur. Les agents sont en liaison directe et envoient leurs messages directement et explicitement au destinataire. La seule contrainte est la connaissance de l'agent destinataire. Les systèmes fondés sur la communication par envoi de messages relèvent d'une distribution totale à la fois de la connaissance, des résultats et des méthodes utilisées pour la résolution du problème. Ces systèmes relèvent plus du domaine de l'Intelligence Artificielle Distribuée que celui des SMA ; la structure de la communication est la plus complexe à maintenir parmi toutes, il est très difficile aux agents de savoir l'emplacement des autres dans le cas d'un système ou les agents sont mobiles. De plus, cette structure est peu robuste, elle ne permet pas de réagir à la perte d'un agent ou d'un message.

La Figure 2.6 illustre un échange de messages suivant le mode décentralisé de communication.

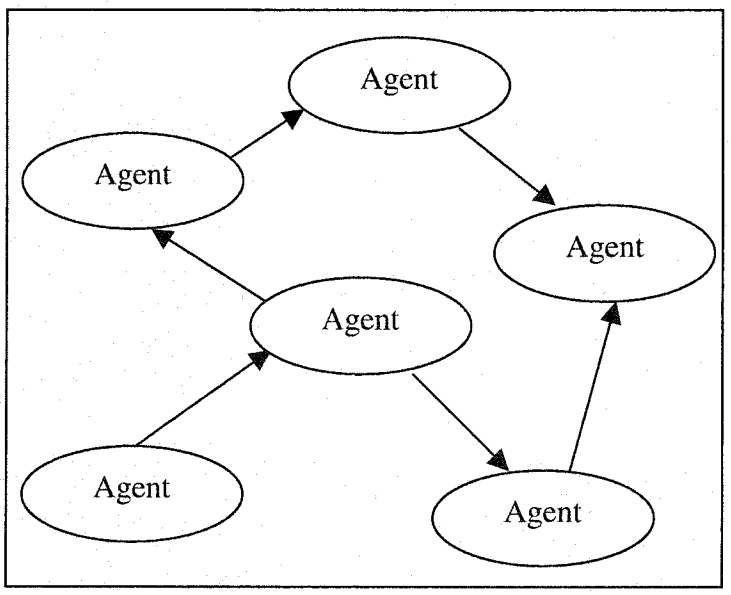

Figure 2.6 Communication décentralisée

Dans le Tableau 2.2, nous résumons les forces et faiblesses des différents schémas de communication présentés ci-dessus. Des valeurs allant de 1 à 3 ont été affectées aux différentes cases pour représenter une performance allant de faible a élevée.

Tableau 2.2 Comparaison des architectures de communication.

Modèle	**Complexité**	**Robustesse**	**Coût des messages (stockage et # d'envoi)**	**Sécurité**	**Total des forces**
Anneau	Moyenne (2)	Moyenne (2)	Elevé (1)	Elevée (3)	8
Etoile	Faible (3)	Elevée (3)	Faible (3)	Faible (1)	10
Hybride	Moyenne (2)	Moyenne (2)	Elevé (1)	Faible (1)	6
Décentralisée	Elevée (1)	Faible (1)	Faible (3)	Moyenne (2)	7

On peut observer, de la simple comparaison présentée au Tableau 2.2, que l'architecture en étoile est le plus favorable si l'on prend en compte le total des forces, et si le SMA implémentant ce modèle de communication n'est pas très sensible vis-à-vis de la sécurité des échanges.

Ontologie

Une ontologie est la spécification explicite d'une conceptualisation (Gruber 1993). Le terme *ontologie* est emprunté à la philosophie où il désigne la théorie de l'existence. Dans les systèmes d'intelligence artificielle, ce qui existe est ce qui peut être représenté.

Dire qu'une ontologie est la spécification d'une conceptualisation signifie en clair qu'une ontologie est une description des concepts et des relations qui peuvent exister pour un agent ou une communauté d'agents.

En ce qui concerne les SMA, une ontologie permet à des agents de :

1. Identifier les concepts et leurs relations avec le monde dans lequel ils évoluent (par exemple : le nom des agents avec lesquels ils vont communiquer) ;
2. Composer des messages avec le vocabulaire du monde afin qu'ils soient compréhensibles par les autres agents.

2.4 Application des systèmes d'agents et multi-agents

Les agents et les systèmes multi-agents sont utilisés dans plusieurs champs d'applications. Ces dernières relèvent de différents domaines, comme par exemple : la gestion des réseaux, la recherche d'information, le commerce électronique et la planification des tâches. Nous présentons dans cette section un échantillon des multiples applications du monde des agents.

2.4.1 Les systèmes d'information coopératifs (SIC)

Les SIC sont généralement caractérisés par la grande variété et le grand nombre de sources d'informations. Ces sources d'informations sont hétérogènes et réparties soit sur un réseau local (Intranet) soit sur l'Internet. De tels systèmes doivent être capables d'exécuter principalement les tâches suivantes :

– la découverte des sources : trouver la bonne source de données pour l'interroger ;

– la recherche d'informations : identifier les informations non structurées et semi-structurées ;

– le filtrage des informations : analyser les données et éliminer celles qui sont inutiles ;

– la fusion des informations : regrouper les informations d'une manière significative.

Le système Warren

Le système multi-agents " Warren " est un système d'agents intelligents pour l'aide des usagers dans la gestion des portefeuilles (Zeng et al., 1996). Ce système combine les données du marché financier, les rapports financiers, les modèles techniques et les rapports analytiques avec les prix courants des actions des compagnies. Toutes ces

informations sont déjà disponibles sur le Web; "Warren" ne fait que les intégrer via des agents spécialisés, les agents d'informations et ensuite les présenter aux usagers. Pour ce faire, "Warren" dispose de six agents ressources, deux agents de tâches et un agent utilisateur pour chaque usager. L'agent utilisateur affiche (via le web) les informations financières de son usager, lui permettant de faire des simulations d'achat et de vente des actions. Il affiche également les prix courants des actions et les nouvelles informations du marché financier. Le même agent permet également d'accéder aux rapports produits par les deux agents de tâches. Ces deux agents fournissent d'une part une intégration graphique des prix et des nouvelles concernant les actions et, d'autre part, une analyse fondamentale des actions en tenant compte de leurs historiques. Les agents d'informations accèdent à différentes sources d'informations, comme les pages Web, les nouvelles de "Clarinet et Dow-Jones", les rapports financiers électroniques de "SEC Edgar" ainsi que d'autres rapports sous un format texte.

Le système NETSA

Le système NETSA est un système multi-agents coopératif, développé à l'université Laval (Côté et al., 1999), et destiné aux environnements riches en informations. Ce système comporte plusieurs types d'agents:

– un agent utilisateur en charge de la cueillette et du filtrage des informations provenant et allant vers l'usager ;

– un agent courtier servant de répertoire pour les agents qui évoluent au sein de NETSA;

– des agents ressources reliés chacun à une ressource d'informations et pouvant rapatrier et mettre à jour les données ;

– un agent d'exécution en charge de la décomposition des tâches et du suivi du déroulement d'exécution des différentes sous-tâches ;

– un agent ontologie en charge du maintien de la cohérence des concepts utilisés par les agents.

2.4.2 Planification d'événements multi-parties

Un exemple d'application de planification basée sur les agents est le prototype mono-agent présenté par Olougouna (2001) pour la planification d'événements multi-parties. Cette architecture permet de planifier une réunion entre un organisateur et plusieurs participants spécifiés par ce dernier. Le prototype permet d'accéder aux calendriers des participants, de déterminer une date libre commune à tous ces derniers et de les notifier de l'événement planifier pour l'enregistrer dans le calendrier. Ceci permet à l'organisateur de le soulager d'une tâche longue et répétitive consistant à accéder à tous les calendriers et les parcourir un à un pour identifier une date convenable à un événement précis. L'architecture se compose de trois modules principaux :

- Un agent AMPE (agent mobile planificateur d'événements) : cet agent inclut une interface usager, des interfaces d'accès à un chercheur d'information et à un identificateur de date. Ces interfaces ou « proxy » garantissent la portabilité des APIs d'accès aux calendriers.
- Un agent MPE (maître planificateur d'événements) : c'est un agent qui réside au serveur des calendriers des participants et constitue le cerveau logique des opérations de planification. Il est programmé pour supporter l'interfaçage avec les calendriers grâce aux APIs offerts par le serveur.
- Une passerelle de communication : elle sert de pont de communication entre les interfaces « proxy » et leurs entités réelles implantées au niveau du MPE.

2.4.3 Application des SMA aux télécommunications

Les télécommunications ont introduit une conception de services décentralisée dans le contexte du Web, créé de nouveaux services de médiation tels que les portails et engendré l'apparition de nombreux fournisseurs de services réseaux qui ne disposent pas de leurs propres services réseaux (Bourron et al., 2001). L'obtention de tels services décentralisés ne peut, bien entendu, être obtenue que grâce à des logiciels pour lesquels les données et le contrôle sont forcément répartis. De ce fait, il est clair que les SMA semblent convenir aux télécommunications. C'est pourquoi les principaux acteurs de

télécommunications mènent actuellement d'intenses activités de recherche sur la technologie agent : British Telecom, France Télécom, Deutch Telekom, NTT, Nortel, Ericsson, Siemens, etc.

2.4.4 Application SMA dans la télé-médecine GUARDIAN

Le système GUARDIAN (Roth et al., 1989) a pour but de gérer les soins aux patients d'une unité chirurgicale de soins intensifs. Les principales motivations de ce système sont: premièrement, le modèle des soins d'un patient dans une unité de soins intensifs est essentiellement celui d'une équipe, où un ensemble d'experts dans des domaines distincts coopèrent pour organiser les soins des patients; deuxièmement, le facteur le plus important pour donner de bons soins au patients est le partage d'informations entre les membres de l'équipe de soins critiques. Particulièrement, les médecins spécialistes n'ont pas l'opportunité de superviser l'état d'un patient minute par minute; cette tâche revient aux infirmières qui, quant à elles, ne possèdent pas les connaissances nécessaires à l'interprétation des données qu'elles rassemblent.

Le système GUARDIAN répartit donc le suivi des patients à un certain nombre d'agents de trois types différents. Les agents *perception/action* sont responsables de l'interface entre GUARDIAN et le monde environnant, établissant la relation entre les données des senseurs et une représentation symbolique que le système pourra utiliser, et traduisant les requêtes d'action du système en commandes pour les effecteurs[1] . Les agents en charge du raisonnement sont responsables d'organiser le processus de prise de décision du système. Finalement, les agents en charge du contrôle (il n'y en a habituellement qu'un seul) assurent le contrôle de haut niveau du système.

2.5 Synthèse sur les SMA

La technologie agent et multi-agents n'est pas un concept voué à rester un simple sujet de recherche en laboratoires. Plusieurs exemples d'applications existent déjà, quelques-unes d'entre elles ont été présentées dans ce chapitre. Il est toutefois difficile

[1] Traduction du terme anglais *effectors*.

de concevoir et de bâtir un système multi-agents. En effet, la conception de ce type de système comporte toutes les difficultés inhérentes aux systèmes répartis, auxquelles s'ajoute le caractère flexible et compliqué des interactions entre agents.

Une des grandes raisons de l'intérêt que connaissent les SMA est l'Internet, où la population d'agents est sans cesse grandissante. Ces agents devront dans l'avenir savoir collaborer et se coordonner afin de réaliser les buts que leur ont dessinés leurs créateurs. Dans ce type d'environnement, les agents rencontrent deux défis majeurs: ils doivent être en mesure de se rencontrer et inter-opérer. Une première solution à ce problème est l'introduction d'agents intermédiaires (*brokers*, *facilitateurs*, etc...) (Côté et al., 1999 ; Troudi et al., 1999). Ces agents intermédiaires ont pour principales fonctions :

1. d'associer au mieux les besoins des utilisateurs et les services des fournisseurs ;
2. d'unifier et de traiter les réponses des fournisseurs pour produire un résultat approprié ;
3. de notifier les utilisateurs de chaque changement d'informations.

Dans cette optique d'interopérabilité, un grand nombre de langages de communications pour les agents ont été développés, basés pour plupart sur la théorie des actes du discours. Bien que les performatives offertes par ces langages permettent de caractériser les types des messages, ils ne permettent pas encore aux agents de comprendre explicitement les concepts "discutés". Le problème des ontologies reste donc ouvert.

Un autre problème critique est l'allocation de ressources limitées à un bon nombre d'agents. Des mécanismes basés sur les principes économiques ont été proposés pour résoudre ce problème. Dans de telles approches, on suppose que les agents sont égo-centrés et ne cherchent qu'à maximiser leur utilité. Les sous-domaines où on a appliqué ces principes économiques sont l'allocation de ressources, l'allocation de tâches et la négociation. Là aussi, la recherche n'a fait qu'entrevoir des techniques rudimentaires basés sur les lois du marché et il reste bien des domaines à couvrir comme par exemple les ventes aux enchères, les comportements acheteur(s) vendeur(s), le partage du marché, etc.

Le domaine des SMA demeure encore aujourd'hui un domaine rempli de défis à surmonter, donc très ouvert pour la recherche. Le développement des applications basées sur les SMA nécessite notamment des recherches approfondies et il convient en particulier :

- de concevoir plus de méthodes et d'outils pour faciliter l'implantation et la construction des systèmes à agents collaboratifs ;
- de maîtriser et de bien identifier la coordination entre les agents. Il s'agit en particulier d'établir une théorie claire et formelle pour cette coordination ;
- d'assurer les critères de stabilité, de clarté et de performance pour de tels systèmes ;
- de trouver des techniques qui permettent l'évaluation, la vérification et la validation de ces systèmes.

CHAPITRE III

ARCHITECTURE DE PLANIFICATION ÉLECTRONIQUE D'ÉVÉNEMENTS

De plus en plus, les applications à base d'agents vont apparaître pour remplacer l'usager dans l'accomplissement des tâches répétitives ou coûteuses en temps. Cette automatisation grâce aux agents mobiles a été un domaine très actif en recherche. Une application en particulier a suscité un grand intérêt, il s'agit de la planification de réunions ou d'événements électroniques. Cette dernière nécessite beaucoup de temps et d'efforts répétitifs pour l'échange d'information entre les individus concernés. Les multiples solutions basées sur les agents relevaient bien plus du domaine de l'intelligence artificielle que des agents mobiles, surtout celles utilisant une approche multi-agents. Elles visaient les problèmes complexes de négociation inter-agent et proposaient plusieurs protocoles comme solution. Cependant, un haut niveau d'intelligence pour les agents atténue le gain en terme de temps de traitement et défavorise l'approche agent par rapport aux approches conventionnelles. Nous proposons dans ce chapitre une nouvelle approche multi-agents ne requérant aucune intelligence au niveau des agents pour la planification d'événements électroniques. Nous allons explorer les motivations de cette approche puis décrire l'architecture développée et ses principales composantes. Par la suite, un algorithme de coordination ainsi qu'une analyse de complexité de ce dernier seront présentés. Finalement, les détails de la composante de communication seront mis en évidence.

3.1 Motivations et bénéfices prévus

Il existe en ce moment plusieurs logiciels permettant de planifier des rendez-vous. L'automatisation de cette planification par les agents mobiles a suscité bien des travaux de recherche. Un travail en particulier a proposé un modèle mono-agent se

servant des calendriers du logiciel *MS Outlook* pour effectuer la planification à la place de l'organisateur (Olougouna, 2001). Dans ce qui suit, nous traiterons les motivations de l'utilisation d'un système multi-agents et les avantages que nous attendons de ce dernier.

3.1.1 Motivations

La solution client/serveur pour la planification d'événements multi-parties présente, comme mentionné au chapitre précédent, plusieurs inconvénients. Le temps d'exécution de la tâche, la charge que cette dernière induit sur le réseau ainsi que la taille de mémoire qu'elle requiert pour télécharger les calendriers de tous les participants sont autant de ces inconvénients qui justifient le choix d'une approche basée sur les agents mobiles.

Ainsi, plusieurs travaux sur l'alternative basée sur les agents ont vu le jour (Ashir et al., 1997; Garrido et al., 1996; Jeong et al., 1999). Ces derniers portent majoritairement sur les mécanismes de décision et l'intelligence nécessaires aux agents pour effectuer la tâche. Ceci dit, dans la mesure où le problème de planification est défini comme la recherche d'un compromis sur la date et le créneau temporel convenables à tous les membres concernés, il est plus pertinent de mettre l'accent sur une architecture à base d'agents permettant la réalisation de cette tâche et ainsi éviter les inconvénients de l'intelligence au niveau des agents. Glitho et al. (2002) ont présenté une application autonome utilisant un modèle mono-agent; du point de vue de la performance, cette dernière est légèrement meilleure qu'une version améliorée de l'approche client/serveur.

Souvent, les événements à planifier impliquent des gens travaillant dans des départements différents d'une même compagnie ou même dans des compagnies différentes. Ceci implique que leurs calendriers résident sur des serveurs différents. L'approche mono-agent propose un agent qui fait le parcours de tous les serveurs, cette migration affecte évidemment le gain en temps. Un modèle multi-agent propose un scénario différent : les agents sont envoyés à raison d'un agent par serveur ou groupe de serveurs, ainsi le temps d'exécution devrait être réduit.

3.1.2 L'approche multi-agents et les avantages escomptés

Dans le cas de la planification multi-parties d'événements électroniques impliquant plusieurs serveurs de calendriers, il existe deux scénarios uniquement. Le premier consiste à envoyer un agent par serveur; le deuxième consiste à affecter un agent par groupe de serveurs, chaque agent faisant bien sûr le tour de tous les serveurs de son groupe. Ce dernier scénario ne sera cependant pas pris en compte pour deux raisons : premièrement parce qu'il combine les deux approches mono-agent et multi-agents ensemble et ne permet pas ainsi de dissocier les avantages de l'une par rapport à l'autre; deuxièmement, le nombre de serveurs considérés par ce mémoire n'étant pas élevé, ce scénario paraît peu pertinent.

Ce mémoire se focalisera donc sur la conception d'un système multi-agents impliquant un agent par serveur de calendrier. Pour illustrer la planification d'un événement à l'aide de l'approche multi-agents, supposons qu'un usager *A* veut planifier un rendez-vous avec les usagers *B, C, D, E* et *F*. les usagers *A* et *B* sont dans le même domaine, leurs calendriers résident sur le serveur 1. Les calendriers de *C* et de *D* sont sur le serveur 2, ceux de *E* et *F* sur le serveur 3. La planification se fait selon le déroulement suivant :

- L'usager *A* lance l'agent client, à travers l'interface usager, il spécifie la durée de la réunion et la période dans le temps pour la planifier; il rentre aussi les adresses des participants incluant lui-même et leurs serveurs de calendrier.
- L'agent client génère les agents planificateurs avec les arguments de départ (date et période pour la réunion, le serveur vers lequel ils doivent migrer, les participants qui leur sont assignés).
- L'agent client envoie les agents planificateurs vers les serveurs 1, 2 et 3.
- Les agents planificateurs accèdent aux calendriers et communiquent avec l'agent client, qui les coordonne pour définir la date de la réunion.
- L'agent client avertit les agents planificateurs pour notifier les participants en cas de réussite.

- Après s'être assuré de la notification de tous les participants, où en cas d'échec, l'agent client ordonne aux agents planificateurs de se supprimer et se supprime à son tour.

La Figure 3.1 illustre le déroulement de la planification qui vient d'être décrit ci-dessus.

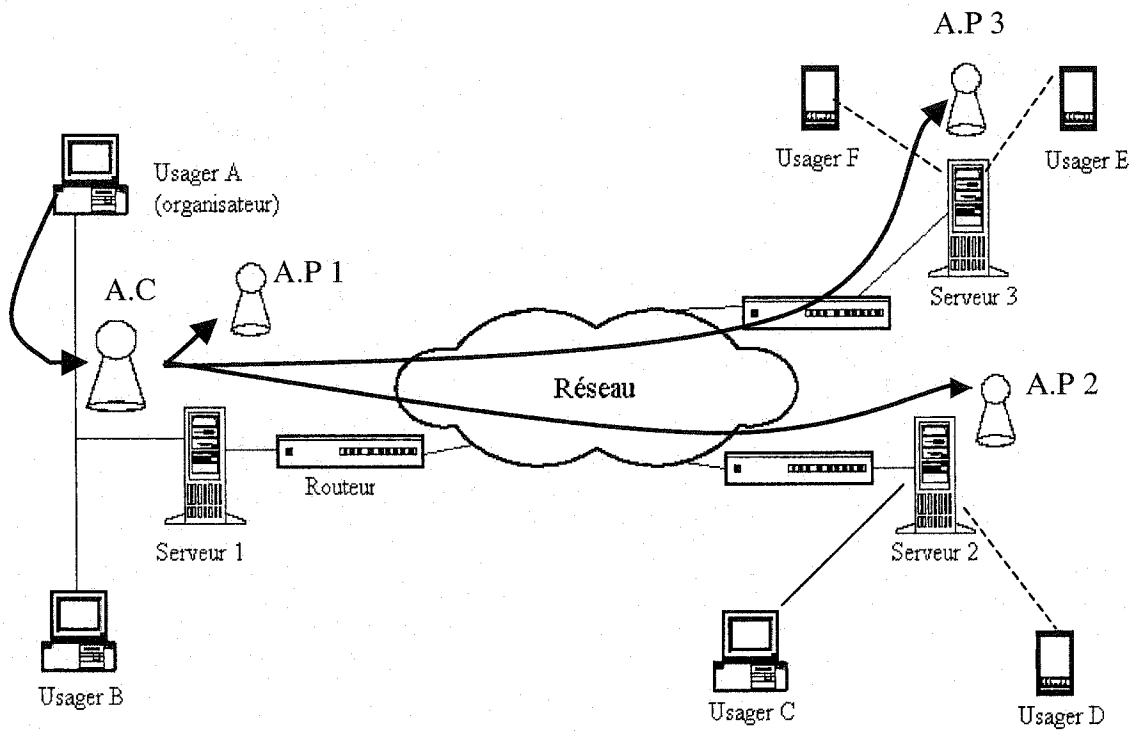

Figure 3.1 Scénario de planification multi-agents

L'adoption de l'approche multi-agents peut présenter plusieurs avantages. Parmi ces derniers, il y a d'abord ceux qu'il partage avec l'approche mono-agent et qui sont les bénéfices propres au paradigme agent mobile :

- la réduction de la charge sur le réseau et le temps de traitement, étant donné que les calendriers ne sont plus transférés mais consultés à leurs emplacements respectifs ;

- l'automatisation de la tâche et la confidentialité de l'information personnelle puisque, d'une part, aucune intervention de l'utilisateur n'est exigée lors de la planification et que, d'autre part, les accès aux calendriers ne sont plus partagés.

En plus de ces avantages, on s'attend du système multi-agents qu'il présente une meilleure performance que celle de l'alternative mono-agent, et cela grâce à la coopération ayant lieu entre les multiples agents. En effet, le fait de se partager les serveurs devrait permettre un gain significatif sur le temps nécessaire à la tâche, ce gain résulterait de l'élimination du temps de migration inter-serveurs que nécessite une approche mono-agent. De plus, une perte d'un ou de plusieurs agents n'affecte pas l'exécution de la planification puisque, dans l'architecture multi-agents, l'agent central (Agent Client) qui réside sur le poste de l'organisateur garde constamment un historique du déroulement de la tâche. Ainsi, en cas de perte d'agents, l'agent central en régénère de nouveaux, tout en leur transmettant les arguments qui leur sont nécessaires pour poursuivre la planification à l'état où leurs prédécesseurs l'ont laissé.

Un dernier avantage du recours à un système multi-agents est la possibilité d'établir un mécanisme de négociation entre les agents. Ce mécanisme permettrait d'atteindre un consensus sur une période, même si cette dernière ne s'affichait pas libre pour tous les participants. Ceci pourrait éventuellement se faire en déplaçant certains événements jugés moins importants ou en négligeant certains participants. Cet aspect du système multi-agents ne sera pas toutefois considéré dans ce mémoire, il fera l'objet de travaux futurs.

3.2 Système multi-agents

Dans cette section, nous présenterons d'abord les hypothèses émises lors de la modélisation, ensuite nous introduirons les deux composantes principales du système multi-agents (SMA) : l'unité centrale appelée *agent client* et l'unité de planification ou *agent planificateur.*

3.2.1 Hypothèses

L'architecture multi-agents s'appuie sur les hypothèses suivantes :

- Une interface d'accès au calendrier existe sur les serveurs de calendriers. Cette hypothèse est tout à fait réaliste étant donné qu'une telle interface a déjà été introduite (Olougouna, 2001).
- Les serveurs de calendriers offrent une plate-forme d'agents (Grasshopper). Cette plate-forme permettra d'accueillir les agents mobiles et d'exécuter leurs codes.
- Le langage de programmation utilisé pour l'implémentation de l'architecture multi-agents est Java, car ce dernier supporte la portabilité et représente le langage le plus utilisé pour les plates-formes d'agents mobiles.

3.2.2 Architecture des unités du SMA

L'architecture multi-agents proposée est un type de système à tableau noir, elle repose sur un modèle centralisé de partage d'information. Ce choix se justifie par le faible coût de communication et la simplicité de ce dernier, ces avantages ont été présentés au chapitre précédent. Les unités clés de l'architecture multi-agents sont l'agent client et l'agent planificateur.

A. L'agent client

La Figure 3.2 illustre l'architecture de l'agent client et présente un schéma des interactions survenant entre ses composantes. Cet agent est lancé par l'organisateur et réside sur sa machine tout le long de la planification. Sa tâche principale est de coordonner les agents planificateurs après les avoir générés et transférés aux serveurs de calendriers. Les principaux modules de cet agent sont décrits dans les paragraphes suivants.

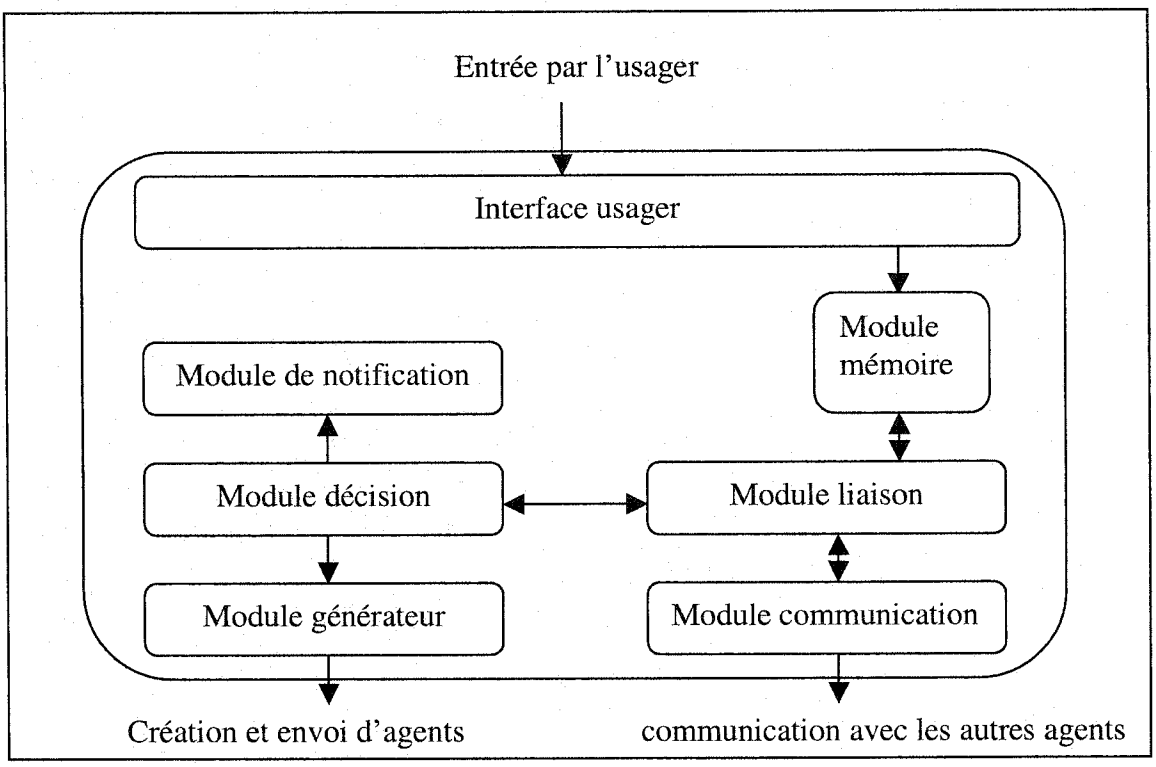

Figure 3.2 Architecture de l'agent client

Interface usager

Cette entité constitue le point d'entrée à la planification multi-agents, elle interagit avec l'organisateur de l'événement pour recevoir les données nécessaires à la tâche. Ces dernières sont :

- la liste des participants, et leur adresses électroniques ;
- les données concernant la réunion (durée, période, ...) ;
- les préférences de l'usager.

Module mémoire

Le premier rôle de ce module est d'enregistrer les données transmises par l'organisateur à l'interface usager au début de l'exécution. Son second rôle est de garder un historique du déroulement de la planification. Ceci aidera à détecter tout

dysfonctionnement tel la perte d'un agent ou d'un message. À cette fin, en plus des informations reçues de l'interface usager, le module devra recevoir de la part d'autres modules et retenir les données suivantes :

- l'identification des agents envoyés sur le réseau ;
- l'état des agents envoyés (actif ou inactif) ;
- le contenu des messages échangés par les agents ainsi que l'information sur ces messages (provenance, sujet).

Afin de permettre une analyse du déroulement de la planification, le stockage des données doit se faire selon un format bien défini. Ce format doit occuper peu de mémoire, être simple à parcourir tout en gardant toutes les informations nécessaires.

Module liaison

Pour garantir le bon déroulement de la planification, il faut vérifier la réception des messages par les agents destinataires et s'assurer que les agents envoyés soient toujours actifs et fonctionnent comme prévu. Ces tâches incombent au module liaison. Pour les accomplir, il faudra analyser le contenu des messages échangés et les transmettre dans un ordre temporel au module mémoire. Le contenu des messages permet de détecter l'absence de réponse de la part d'un agent. Dans un tel cas, il faudra à partir de l'historique des messages déterminer si cette absence est due à la perte d'un agent ou à la non-réception d'un message. Le module liaison doit communiquer tout événement au module décision et l'enregistrer au module mémoire.

Module de communication

Le module de communication a la responsabilité de rendre la communication possible entre les différents agents. Pour cela, il devra savoir l'emplacement de tous les agents. Aussi, dans le contexte où les agents appartiennent à des plates-formes différentes et utilisent des protocoles de communication différents, le module de communication doit faire la conversion de ces protocoles. Dans notre cas, les agents utilisent la même plate-forme (Grasshopper), le module de communication s'appuie

donc sur le service de communication de cette dernière qui utilise une entité intermédiaire appelée *proxy*. Ce dernier garde une référence de l'emplacement du serveur sur lequel réside l'agent destinataire. Ainsi, pour communiquer avec un autre agent, il suffit de fournir son identification au module de communication.

Module de notification

Pour le cas de l'agent client, ce module reçoit du module de décision un message indiquant l'échec de la planification. Étant donné que ce dernier réside sur la machine de l'organisateur, il peut l'aviser à l'aide d'un message instantané. Ce message peut indiquer des informations sur les causes de l'échec, afin de permettre éventuellement la relance d'une autre planification.

Module de décision

La tâche de planification doit suivre un schéma précis, le rôle de chaque agent de planification doit être bien défini et une coordination entre les différents agents doit être établie et gérée par une unité centrale. Le module de décision joue le rôle de cette unité coordonnatrice. Il implémente un algorithme de coordination lui permettant, entre autres, de prendre des décisions en ce qui concerne le remplacement d'agents perdus par la création de nouveaux, mais aussi sur la réussite ou l'échec de l'accord sur une date. Cet algorithme est présenté dans les paragraphes suivants. Ce dernier peut être amélioré pour considérer les préférences de l'usager et pour établir une négociation sur une date.

Module générateur

Cette unité implémente le processus de création d'un agent, elle invoque le constructeur d'un agent et lui transmet les paramètres d'entrée avant de l'envoyer à la destination voulue.

B. L'agent planificateur

L'agent planificateur est plus léger étant donné qu'il se déplace sur le réseau. Sa tâche principale est d'accéder aux calendriers sur le serveur afin d'en analyser le contenu et de le transmettre à l'agent client. Cet agent ne profite d'aucun mécanisme de décision et agit sous l'influence de l'agent client. L'architecture de l'agent planificateur est présentée à la Figure 3.3. Les différents composantes de cette architecture sont : le module de communication, le module de notification, le module d'accès au calendrier, et le module de planification.

Module de communication

De la même manière que l'agent client qui doit envoyer des requêtes à l'agent planificateur, ce dernier a besoin de communiquer ces réponses. Son module de communication occupe les mêmes fonctions que pour l'agent client, à la seule différence que pour l'agent planificateur il y a un seul destinataire étant donné que la communication suit un modèle de tableau noir.

Module de notification

Pour le cas de l'agent planificateur, le module de notification doit aviser les participants qui lui sont assignés en cas d'accord sur une date. Le message doit contenir la période et l'objet de la réunion, ainsi que l'identification des autres participants. Dans cette architecture, le module de notification implante un mécanisme d'envoi de courriers électroniques, étant donné que ce dernier est le média adopté pour la notification.

Module d'accès au calendrier

Ce module interagit avec le serveur de calendrier. Il identifie l'emplacement des agendas des participants et sert d'interface pour ouvrir le contenu de ces agendas et le traduire en données manipulables en Java. Ces données sont ensuite transmises au module de planification. Ceci se justifie par le fait que les APIs permettant l'accès aux calendriers ne sont pas toutes écrites en Java.

Module de planification

Le module de planification permet d'analyser et de traiter le contenu des agendas des participants qui résident sur un même serveur. Son rôle consiste à transmettre les périodes libres communes à tous les participants au module de communication qui les communiquent à son tour à l'agent client. Il doit aussi transmettre, le cas échéant, la période prévue pour l'événement et les adresses de courriers des participants au module de notification.

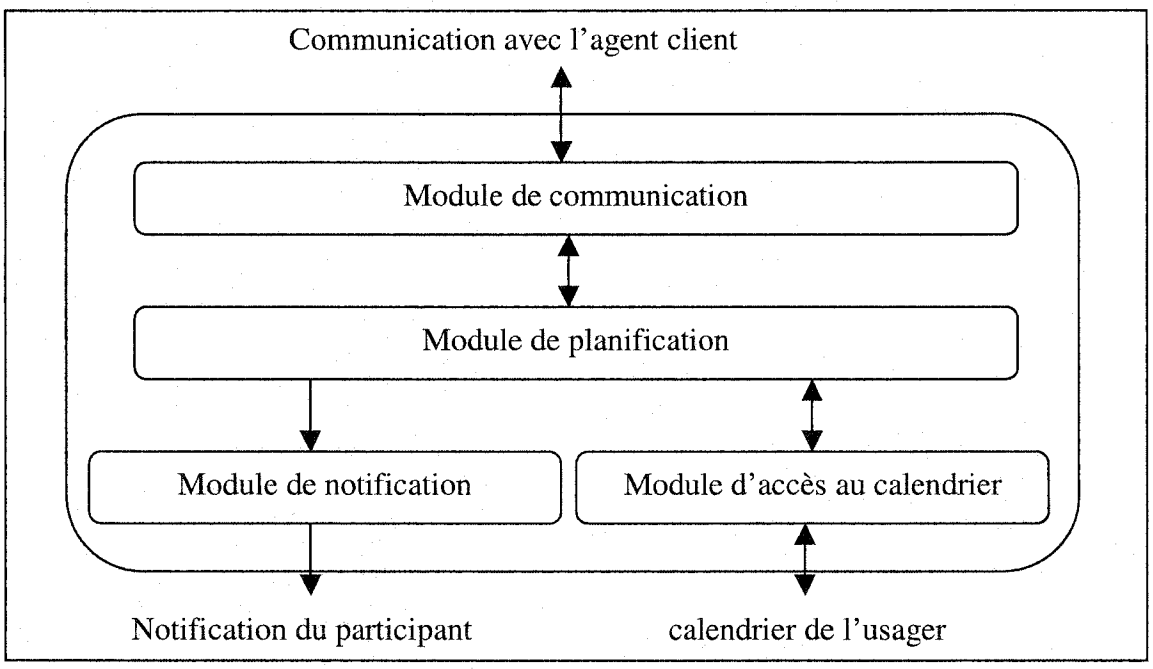

Figure 3.3 Architecture de l'agent planificateur

3.2.3 Algorithme de coordination

La tâche de planification à l'aide d'un SMA doit suivre une procédure bien définie. Pour cela, le SMA doit utiliser un algorithme de coordination pour combiner les résultats obtenus par les différents agents et minimiser ainsi le temps d'exécution. La Figure 3.4 présente l'algorithme utilisé pour la planification par le SMA. Cet algorithme est implémenté au niveau du module de décision de l'agent client. Une analyse de l'algorithme est présentée dans les sections suivantes.

Initialisation :
adresses usagers : $(U_1,\ldots,U_k)$.
serveurs : (S_1:$\{U_1,\ldots,U_r\}$,……, S_i : $\{U_s,\ldots,U_t\}$, S_n : $\{U_{t+1},\ldots,U_k\}$).
période : $P = [\ J_{début}\ ,\ J_{fin}\ ]$.
durée : D.
première date libre : $PDL = J_{début}$.
réussite : = FAUX.
temps maximal d'exécution : = T_{max} .

Début
POUR i : = 1 **À** n
Créer agent planificateur AP_i (D, PDL, J_{fin})
Envoyer AP_i vers S_i
Fin **POUR**
TANT QUE (réussite = FAUX) **ET** ($PDL \leq J_{fin}$) **ET** ($t \leq T_{max}$)
trouver_date : = 1
i : = 1
Réclamer les PDL_i des AP_i en //
TANT QUE ($i \leq n$)
SI AP_i a retourné PDL_i **ALORS**
SI ($i \neq 1$) **ALORS**
SI ($PDL_i < PDL_{i-1}$) **ALORS**
PDL : = PDL_i
SI ($PDL_i = PDL_{i-1}$)
trouver_date : = trouver_date + 1
SINON
PDL : = PDL_1
Fin **SI**
i : = i + 1
SINON AP_i a été perdu
Créer un nouveau AP_i (D, PDL, J_{fin}) et l'envoyer à S_i
Fin **SI**
Fin **TANT QUE**
SI (trouver_date = n) **ALORS**
Réussite : = VRAI
Fin **TANT QUE**
SI (réussite = VRAI) **ALORS**
POUR i : = 1 **À** n
Réclamer à AP_i de notifier tout $U \in S_i$ d'une réunion à PDL
Ordonner à AP_i de se supprimer
Fin **POUR**
SINON (échec de communication ou de consensus sur une date)
Notifier l'organisateur U_1 de la nature de l'échec.
Fin **SI**
FIN

Figure 3.4 Algorithme de coordination

Analyse de l'algorithme

L'algorithme de coordination proposé est de type déterministe, c'est-à-dire qu'il fonctionne jusqu'à l'obtention d'une réussite ou d'un échec de la planification. Il intègre deux boucles dont une principale qui renferme une autre boucle. La première boucle crée tout simplement un agent pour chaque serveur avec les paramètres de départ de la planification. La deuxième et principale boucle s'exécute jusqu'à l'identification d'une période libre pour la réunion ou le constat d'un échec. Elle comprend une autre boucle qui collecte et compare les PDL (première période libre commune dans l'agenda des participants affectés à un agent), pour chaque AP. L'identification de la réussite est faite grâce à la variable *réussite* qui devient VRAI au cas où tous les PDL ont la même valeur; l'échec, quant à lui, est détecté lors du dépassement de la limite de la période de planification (J_{fin}).

Un autre paramètre T_{max} est ajouté pour contrôler l'exécution des instructions. Ce dernier représente le temps maximal de fonctionnement pour l'algorithme. Il permet d'éviter des itérations infinies au cas où un AP ne réussirait pas à communiquer avec l'AC à cause de problèmes sur le réseau.

Hypothèses

Afin de déterminer la complexité ainsi que la performance de l'algorithme, il faut d'abord définir les paramètres qui influencent sa performance et émettre des hypothèses concernant ces derniers. L'algorithme de coordination est gouverné par trois variables, la première et la plus déterminante pour notre comparaison étant ***n*** le nombre de serveurs de calendrier, la deuxième étant ***k*** le nombre de participants à l'événement qui doit être planifié et la dernière étant ***m*** le nombre d'intervalles de temps à analyser; celle-ci s'obtient en divisant la période de planification par la durée de l'événement

$$m = P/D = (J_{fin} - J_{début})/D$$

L'analyse de complexité s'appuie sur les hypothèses suivantes :

- Le nombre d'intervalles ***m*** ne varie pas beaucoup lors des planifications de sorte qu'il peut être considéré comme constant. Cette hypothèse est tout à fait réaliste,

puisque la planification se fait généralement dans une période assez limitée (2 ou 3 semaines) et que le nombre d'heures disponibles pour une réunion dans une journée est constant (8 heures par jour de travail).

- Le nombre de serveurs où logent les agendas des participants n est variable, mais ce dernier est toujours plus petit que m. Le nombre de serveurs est généralement de l'ordre d'une dizaine au maximum. Cependant, m n'est pas très grand en comparaison à n, de sorte que le poids du premier peut être négligé par rapport à celui du second.
- La taille de l'information sur un intervalle d'un agenda envoyée dans un message lors de l'exécution de l'algorithme est égale à $1/m$ fois la taille d'un agenda. Ceci veut dire qu'aucune compression de l'information n'est effectuée, et que l'agent envoie tout simplement une partie de l'agenda.

Efficacité dans le pire cas

En considérant les hypothèses émises auparavant, il faut maintenant considérer le pire scénario pour déterminer l'efficacité de l'algorithme. Le pire cas se présente quand le nombre de participants est égal au nombre de serveurs ($n=k$, un agenda par serveur de calendrier). En effet, à ce moment là, le nombre de messages échangés et d'itérations effectuées est égal à un par participant, au lieu d'être égal à un par groupe de participants et de fait par groupe d'agendas.

Toujours dans le pire cas, en ce qui a trait aux périodes libres dans les agendas des participants, le premier intervalle libre commun à tous se trouve à la fin de la période définie pour la planification, ce qui veut dire que l'exécution de l'algorithme se fait sur les m intervalles en entier. De plus, un intervalle sur deux est libre pour chaque participant, ce qui revient à dire que les agents envoient un message une fois à chaque deux lectures de l'agenda.

En considérant tout ce qui vient d'être écrit, dans le pire cas, l'algorithme effectue au départ n itérations pour créer les AP et ensuite rentrer dans une boucle qui s'exécute sur tous les intervalles de temps; donc, il effectue m itérations tout en

considérant que chacune de ses itérations enclenche une boucle qui est répétée n fois, car la procédure demande à tous les agents sur les serveurs d'envoyer leurs PDL. Donc, le temps d'exécution nécessaire au pire cas est :

$$f(n) = n + m * n = (m+1) * n.$$

En conclusion, l'algorithme réagit de façon linéaire par rapport à n, le nombre de serveurs de calendriers, ce qui représente la meilleure complexité possible.

En terme de performance par rapport à la charge de données échangées, en prenant en compte la troisième hypothèse stipulant que le taux de compression des données dans les messages est égal à un, on trouve que l'algorithme envoie dans le pire cas un message par deux intervalles de planifications. La Figure 3.5 illustre le fonctionnement de l'algorithme dans le cas décrit. Ceci veut dire que l'algorithme permet à l'approche SMA de produire 50% moins de charge sur le réseau en comparaison à l'approche client/serveur, où on a besoin de ramener toute la période de planification voulue (donc les m intervalles en entier) du coté du serveur. En comparaison avec l'approche mono-agent, il est difficile d'estimer le gain en terme de charge, car cela dépend énormément de la taille nécessaire à un agent pour effectuer la planification. Ceci dit, le fait que plusieurs agents se partagent la tâche de planification en usant d'un algorithme linéaire permet de prédire un meilleur rendement sur le temps d'exécution pour les SMA par rapport à l'approche mono-agent.

Efficacité dans le cas moyen

Dans un cas moyen, il y a en moyenne (k/ n) participants par serveur et la réussite est atteinte au milieu de la planification, ce qui veut dire que l'algorithme effectue uniquement $[n + (m * n)/2]$ itérations. Évidemment, la complexité n'est pas améliorée par rapport au pire cas. Cependant, le nombre de messages échangés pour un même nombre d'agendas est multiplié par le nombre (n/k), car chaque agent envoie un intervalle par groupe d'agendas qui lui sont affectés. Donc, la charge de données échangées est améliorée de $0.5*(n/k)$ par rapport à l'approche client/serveur.

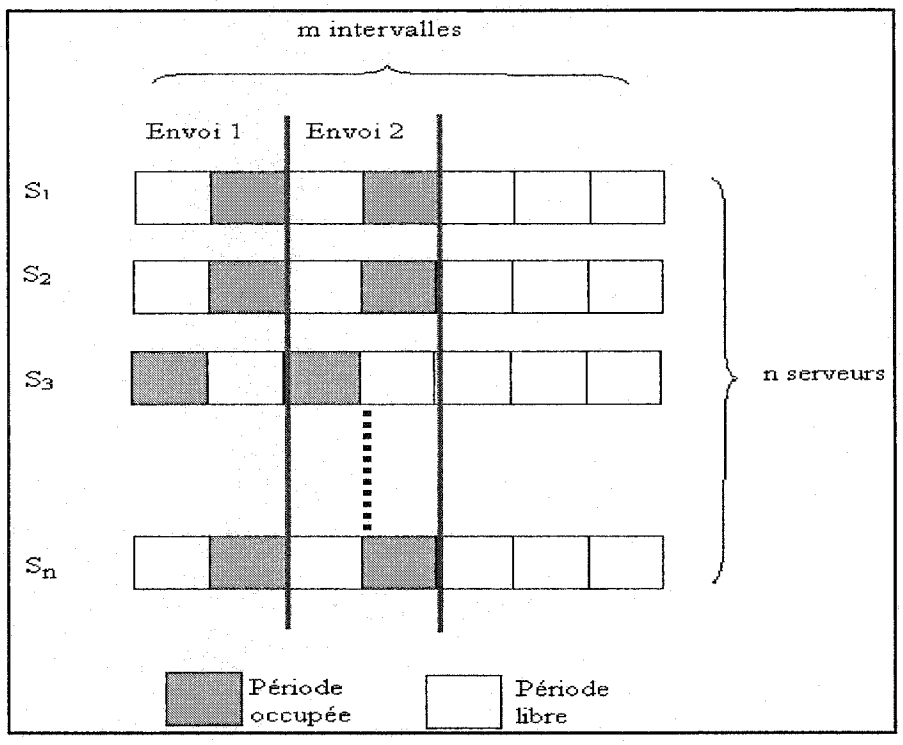

Figure 3.5 Nombre de messages échangés par l'algorithme de coordination dans le pire cas de planification

3.2.4 Communication SMA

Le service de communication est l'aspect le plus essentiel de l'architecture SMA : sans lui, même un minimum de coordination ne pourrait être assuré. Dans cette section, nous discutons de la nature de la communication fixée pour le SMA, ainsi que le caractère des messages échangés permettant de donner un sens aux interactions inter-agents lors de la planification.

Nature de la communication

Étant donné que l'architecture SMA proposée pour la planification répartie d'événements multi-parties contient un organe centralisateur (l'agent client), et que ce dernier doit collecter les informations reçues des différents agents de planification

s'exécutant sur les serveurs de calendriers, le type le plus approprié pour la communication est le mode asynchrone. En effet, le mode asynchrone de communication, contrairement au mode synchrone, ne bloque pas l'agent central en attente de la réception d'une réponse d'un agent du système. Ce détail est très important car la perte d'un message provenant d'un AP ou la perte d'un AP lui-même ne doit pas affecter le fonctionnement de la planification SMA, et ce, pour respecter la condition de robustesse.

L'agent central de l'architecture SMA doit effectuer plusieurs tâches en parallèle, il doit aussi associer chaque message à sa provenance et être capable de recevoir plusieurs réponses à la fois. Tout ceci peut être accompli en utilisant le mécanisme de type *notification* du mode de communication asynchrone. L'agent central crée un objet de type écouteur (*listener*) pour chaque agent avec lequel il communique. L'écouteur va servir en même temps d'identificateur au message et de notifiant de la réception de ce dernier, évitant ainsi que l'agent central n'attende chaque fois pour une réponse.

Format des messages

Le fait d'établir une communication entre les agents ne permet pas à ces derniers de partager leurs connaissances. Pour donner un sens aux interactions des agents du SMA et leur permettre de comprendre le contenu des messages qu'ils se communiquent, il faut définir un format pour les messages échangés. Le choix de ne pas utiliser des protocoles existants d'échange d'informations entre les agents tels que KQML se justifie par deux arguments principaux.

Premièrement, pour utiliser un protocole tels que KQML, il faut que les modules de communication des agents du SMA puissent supporter ce protocole, ce qui veut dire une augmentation de la taille des agents. Deuxièmement, ces protocoles nécessitent de créer et de maintenir une ontologie, chose qui paraît inutile pour une tâche comme la planification où la nature des actions et des décisions que prennent les agents sont réduits et bien connus.

Ceci dit, le choix de ne pas utiliser KQML pour échanger l'information ne veut pas dire qu'il faut en inventer un nouveau, le format de message retenu s'en inspire et est en réalité une adaptation de KQML pour le SMA présenté dans ce chapitre.

Format du message

La syntaxe des messages dans le SMA ressemble à la forme de base d'un message KQML présentée dans le chapitre précédent. Elle possède un argument en plus, *reply-with* pour un message provenant de l'AC et *in-reply-to* pour un message provenant d'un AP. Ces deux champs permettent au module de décision de distinguer les messages et de détecter l'absence ou des erreurs dans certains d'entre eux. Le principal atout d'un tel format est que tout ce qui est nécessaire à la compréhension du message est inclus dans le message lui-même. La Figure 3.6 illustre un échange entre l'AC et un AP.

Les performatifs

Les performatifs définis pour les messages entre agents représentent le minimum nécessaire à ces derniers pour effectuer la tâche de planification. Ces performatifs sont :

- ask : pour demander le PDL aux différents AP ;
- notify : pour ordonner aux agents AP de notifier les participants qui leur sont assignés ;
- tell : pour transmettre les nouveaux arguments aux agents planificateurs ;
- reply : pour transmettre le PDL à l'AC ;
- remove : pour ordonner aux agents AP de se supprimer.

(**ask :** **:sender** < Agent Client >
:receiver < Agent_planificateur2 >
:language < NULL >
:ontology < NULL >
:content < message4 >
:reply-with < PDL >)

(**reply :** **:sender** < Agent_planificateur2 >
:receiver < Agent Client >
:language < NULL >
:ontology < NULL >
:content < PDL = 7/20 16:00-18:00 >
:in-reply-to < message4 >)

Figure 3.6 Exemple d'échange de messages

CHAPITRE IV

IMPLÉMENTATION ET RÉSULTATS

Pour évaluer la performance du paradigme agent mobile par rapport au client/serveur, le modèle d'agent mobile utilisé a souvent été basé sur des applications mono-agent, négligeant ainsi plusieurs avantages qu'auraient pu amener des applications multi-agents. Par ailleurs, les travaux dans le champ des systèmes multi-agents appartiennent le plus souvent au domaine de l'intelligence artificielle (IA) et cherchent à appliquer les théories de l'IA et à modéliser des comportements tels la coopération, la prise de décision et du contrôle. Au lieu de les confronter expérimentalement aux applications mono-agent et client/serveur en essayant de tirer avantage de l'allocation de tâche et du partage de la charge d'exécution que permet les SMA, la simulation de ces derniers a souvent été utilisée pour comprendre la dynamique des systèmes complexes, et des phénomènes tels celui de la négociation entre entités. L'objectif de ce chapitre est d'implémenter un modèle multi-agents et de développer des prototypes de simulation pour déterminer la performance des SMA par rapport aux systèmes mono-agent; la comparaison entre client/serveur et mono-agent a déjà été faite par Olougouna (2001), et ce, dans un domaine où le client/serveur présente des lacunes par rapport à son alternative agent mobile, en l'occurrence la recherche sur les serveurs d'informations et plus précisément la planification d'événement multi-parties. Ce chapitre débute avec une présentation des prototypes multi-agents et mono-agent développés pour mesurer les apports en performance, il décrit ensuite l'environnement d'expériences et les résultats obtenus pour finir avec une analyse de ces derniers et une synthèse sur l'évaluation.

4.1 Prototypes et environnement expérimental

Avant de commencer à décrire les prototypes, il faut souligner que les corps des programmes des prototypes sont entièrement en langage Java ainsi que la plate-forme des agents qui est Grasshopper. Dans cette section, nous donnons une description des

prototypes multi-agents et mono-agent développés et les modules principaux de ces derniers, pour finir avec l'environnement et le scénario d'expérimentation.

4.1.1 Prototype et scénario multi-agents

Le prototype multi-agents consiste en trois classes et une interface : les classes *AgentClient*, *AgentPlanificateur* et *InterfaceUsager*, et l'interface *IAgentPlanificateur*.

- La classe *AgentClient* : cette classe réalise l'agent central de l'architecture multi-agents, elle hérite de la classe *StationaryAgent* de Grasshopper lui permettant d'avoir les fonctionnalités d'un agent stationnaire et de supporter la communication. Cette classe lance au départ une interface usager pour obtenir les paramètres de la planification. Elle contient une référence au système d'agents local qui offre les méthodes pour invocation à distance et permet de créer des agents. La classe maintient également un proxy de l'agence locale lui permettant de localiser les agences des serveurs de calendriers qui doivent être enregistrées dans un service de domaine (agency domain service). Sachant le nombre de serveurs de calendriers et leurs emplacements, la classe *AgentClient* crée un nombre égal d'agents AP en chargeant le code à partir de la classe *AgentPlanificateur*, elle génère ensuite dynamiquement des proxies en spécifiant l'emplacement des serveurs de calendrier, ces proxies sont des instances de l'interface *IAgentPlanificateur* et supportent la communication asynchrone. La classe *AgentClient* implémente une interface de type *ResultListener* qui lui permet d'être notifié une fois que le résultat d'une communication asynchrone est retourné.
- La classe *AgentPlanificateur* : cette classe contient le code des agents AP de l'architecture multi-agents, elle hérite de la classe *MobileAgent* de Grasshopper et possède ainsi les fonctionnalités d'un agent mobile, en plus de supporter la communication. Elle implémente l'interface *IAgentPlanificateur*, ce qui lui permet d'offrir deux méthodes au service de communication. La classe

AgentPlanificateur contient le code de tous les modules de l'AP en tant que sous-classes, ceci est nécessaire étant donné la nature du processus de migration.

- L'interface *IAgentPlanificateur* : cette interface contient les méthodes des agents AP qui sont accessibles à l'agent AC via le service de communication, elle constitue la base de la génération des proxies des agents AP.

La Figure 4.1 donne un aperçu des deux méthodes offertes par les agents AP à l'agent AC. La première méthode reçoit deux paramètres : une chaîne de caractères représentant le performatif de l'AC et un tableau de chaînes reproduisant le contenu du message. Cette méthode retourne un objet contenant le message de réponse de l'AP. La deuxième méthode reçoit deux chaînes de caractères en paramètre, la première représente l'emplacement du serveur de calendrier vers lequel l'agent AP doit migrer, la seconde indique le nom et l'emplacement des fichiers de calendrier.

```
▪ public Object requestPDL (String performative, String[] contenu)
throws AsyncServerException1;
▪ public void go(String location, String f_name);
```

Figure 4.1 Méthodes de l'AP accessibles via le service de communication

La Figure 4.2 illustre le diagramme des principales classes du prototype multi-agents.

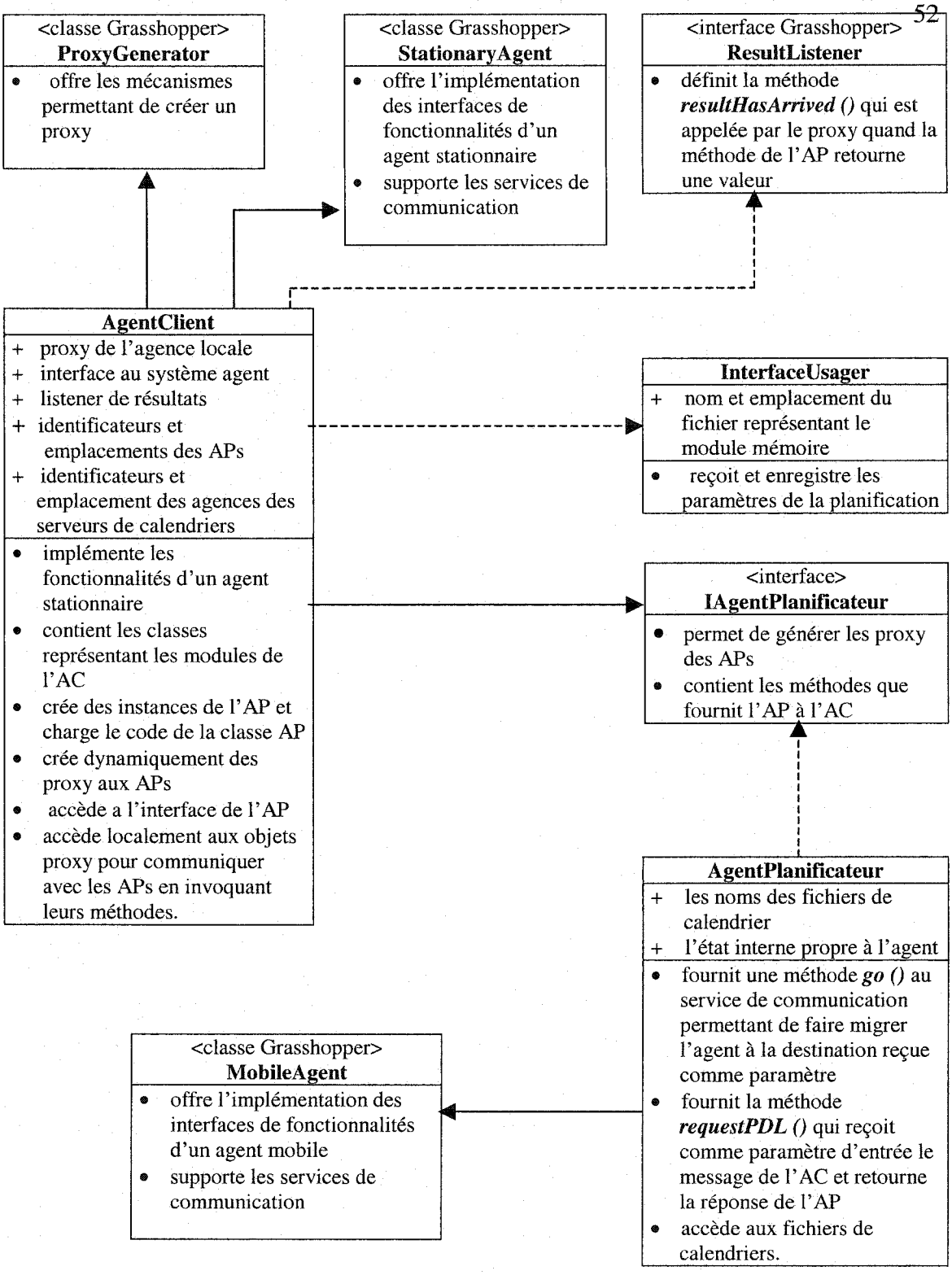

Figure 4.2 Diagramme des classes du prototype multi-agents

Scénario du prototype

Le scénario de planification multi-agents avec trois serveurs de calendriers est illustré à la Figure 4.3, ni les pertes d'agents ni les problèmes de communication n'y sont considérés.

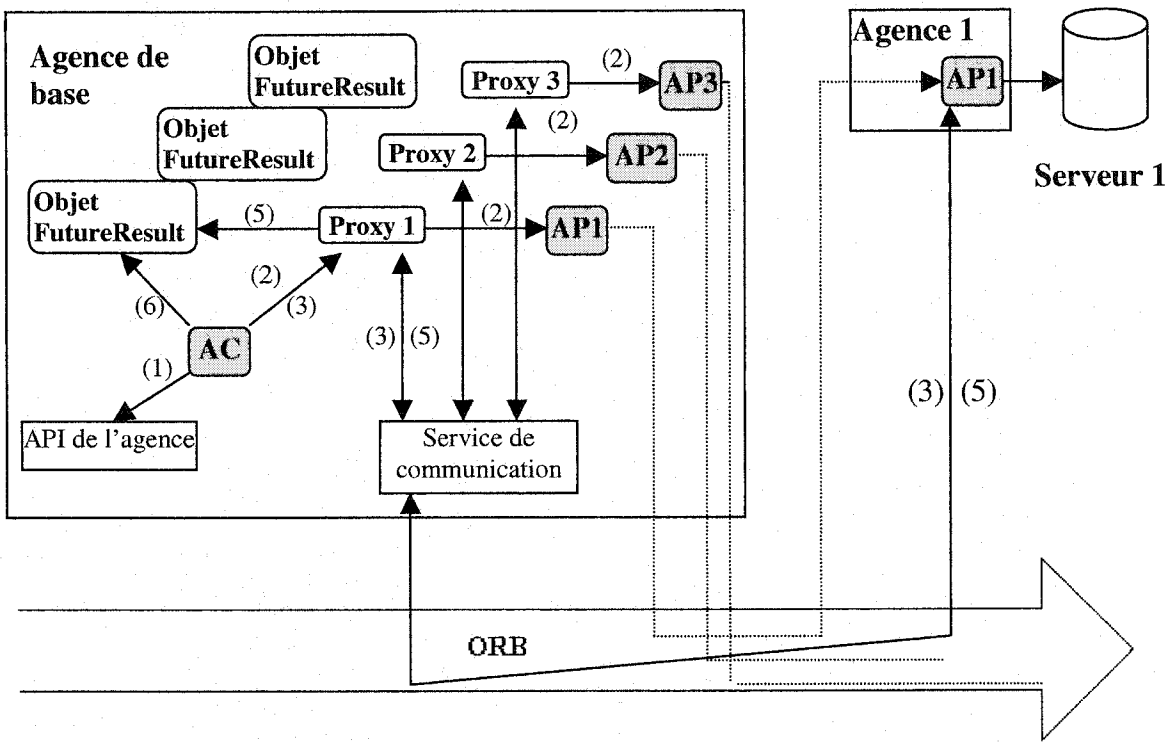

Figure 4.3 Scénario du prototype multi-agents

(1) L'agent AC crée des instances d'agents AP et établit des connexions avec ces derniers via des proxies.

(2) L'agent AC invoque la méthode ***go()*** à travers les objets proxy pour faire migrer les agents vers les serveurs de calendriers.

(3) Les agents AP sont rendus sur les serveurs, l'AC invoque la méthode ***requestPDL()*** avec les paramètres de planification. L'appel de la méthode est transmis par le proxy via le service de communication.

(4) L'agent AC crée un objet *FutureResult* via chaque proxy.

(5) Les agents AP accèdent aux fichiers calendriers et transmettent leurs réponses aux objets proxy qui les inscrivent dans les objets de type *FutureResult*.

(6) L'agent AC est notifié de la réception d'une réponse et retire l'information de l'objet *FutureResult*.

L'agent AC analyse à chaque fois le résultat et invoque de nouveau la méthode ***RequestPDL()*** jusqu'à la fin de la planification.

4.1.2 Prototype et scénario mono-agent

Le prototype mono-agent consiste en une classe unique appelée *MonoAgentPlanificateur* qui hérite de la classe *MobileAgent* de Grasshopper. La classe contient une référence vers le proxy de l'agence locale, ce qui lui permet de localiser les agences des serveurs de calendriers; l'itinéraire suivi par l'agent est défini aléatoirement. La classe *MonoAgentPlanificateur* accède aux fichiers calendriers en utilisant les mêmes interfaces d'accès que celles des classes *AgentPlanificateur* de l'architecture multi-agents.

La Figure 4.4 présente l'algorithme utilisé par le prototype mono-agent pour effectuer la tâche de planification, cet algorithme est de la même granularité que celle de l'algorithme utilisé par le système multi-agents, c'est-à-dire qu'il analyse le calendrier par périodes de temps de la même longueur. Aussi, les paramètres d'initialisation sont les mêmes pour les deux algorithmes. L'ordre de complexité de cet algorithme (en tenant compte des mêmes hypothèses émises au chapitre précèdent) est $O(\boldsymbol{n})$ où $\boldsymbol{n}$ est le nombre de serveurs de calendriers; il est donc du même ordre de complexité que l'algorithme utilisé par le système multi-agents.

À partir de l'algorithme, on peut déduire le scénario utilisé pour la planification mono-agent. En effet, l'agent visite chaque serveur, ensuite il accède au calendrier pour ne garder que les périodes libres en mémoire; à la fin de l'itinéraire, il retourne au serveur de départ et notifie l'organisateur de la première période libre parmi celles enregistrées.

Initialisation :
adresses usagers : $(U_1,\ldots,U_k)$.
serveurs : (S_1:{$U_1,\ldots,U_r$} ,....., S_i: {$U_s,\ldots,U_t$} , S_n: {$U_{t+1},\ldots,U_n$}).
période : P = [$J_{début}$, J_{fin}].
durée : D.
réussite : = FAUX.
POUR i : = 1 **À** m // m = (($J_{début}$ - J_{fin})/ D), est égale au nombre de périodes.
CAL[i] = libre
Fin POUR

1.1 Début
POUR i : = 1 **À** n
Migrer vers le serveur S_i
POUR j : = 1 **À** m
Lire la période j dans le calendrier
SI (période j est occupée)
CAL[j] = occupée
Fin **SI**
Fin **POUR**
Fin **POUR**
Retourner vers le serveur S_1
TANT QUE (réussite = FAUX) **ET** ($i \leq m$)
SI (CAL[i] = libre) **ALORS**
Réussite = VRAI
Notifier U_1
SINON
Incrémenter i
Fin **SI**
FIN

Figure 4.4 Algorithme de planification mono-agent

4.1.3 Environnement d'expérimentation

La machine à partir de laquelle les applications de planification multi-agents et mono-agent sont lancées, ainsi que toutes les machines simulant des serveurs de calendriers, sont dotées de la plate-forme d'agent Grasshopper 2.2.4. Toutes les machines sont des ordinateurs de bureau avec un processeur Pentium III 800 MHz équipés de Windows 2000.

Les calendriers sont stockés en mémoire sous format de fichiers XML. La Figure 4.5 illustre le squelette d'un calendrier XML. Le choix du format est justifié par le fait que le langage XML fournit d'excellents moyens pour décrire les données nécessaires

pour générer un calendrier et présente une solution robuste de plus en plus utilisée par les développeurs d'outils logiciel (Lotus XML Toolkit d'IBM pour ordinateurs de bureau et PDA : http://www.lotus.com/developers/devbase.nsf/homedata/homexmltk).

```
<?xml version="1.0" standalone="yes" ?>
  <!DOCTYPE Year>
- <Year>
  <Yearname>2002</Yearname>

    - <Month>
          <Monthname>January</Monthname>
              - <Day>
                    <Dayname>1</Dayname>
                     - <Hour>
                              <Range>8:00-9:00</Range>
                              <State>libre</State>
                       </Hour>
                     + <Hour>
                              .
                              .
                              .
                     + <Hour>
                 </Day>
               + <Day>
                    .
                    .
                    .
               + <Day>
        + <Month>
                .
                .
                .
         + <Month>
  </Year>
```

Figure 4.5 Modèle de calendrier XML utilisé

Le réseau utilisé durant les expériences est de type Ethernet 100 Mbps, la communication entre les nœuds du réseau se fait par sockets TCP/IP.

4.2 Modèle d'expériences

Dans cette section, nous décrivons les métriques employées et les facteurs considérés pour comparer les performances des architectures multi-agents et mono-agent. Nous décrivons ensuite le plan d'échantillonnage et le niveau de variation des différents facteurs.

4.2.1 Métriques et facteurs de l'expérience

Pour l'évaluation de performance, deux métriques sont utilisées :

1. la *charge réseau*, qui est la mesure de la charge totale générée sur le réseau par la tâche de planification ;
2. le *temps de planification*, qui est la mesure de la durée nécessaire à la planification, et ce, du lancement de l'application jusqu'à l'obtention d'un échec ou d'un succès.

Ces deux métriques sont influencées essentiellement par quatre facteurs :

- le nombre de serveurs de calendriers, n ;
- le nombre de calendriers par serveur, k ;
- la position p du premier créneau temporel libre dans la période de planification ;
- le taux de remplissage des calendriers, c'est-à-dire le nombre d'événements inscrits dans chaque agenda.

La comparaison des performances des systèmes multi-agents et mono-agents se fera selon le scénario du pire cas, c'est -à-dire que le nombre de calendriers par serveur sera égal à 1 et que le taux de saturation des calendriers est de 50% (un événement inscrit au calendrier chaque deux périodes).

4.2.2 Sessions de mesures

Pour effectuer les différentes séances de test, nous avons considéré les conditions suivantes :

- le nombre de serveur de calendriers varie entre 3 et 5 ;
- la durée de planification est de 30 jours ;

- la période de planification est fixe et égale à 1 heure ;
- le nombre de périodes dans une journée est égale à 8 ;
- le succès de la planification comprend trois scénarios : le premier représente le cas où les participants sont disponibles à la première période du premier jour (P = 1); le deuxième scénario représente le cas moyen, la $4^{ème}$ période du 15^{e} jour est la première à être libre pour tous les participants (P = 120); le dernier scénario est celui où le succès n'est atteint qu'au dernier jour de planification (P = 240).

Pour mesurer les performances du prototype SMA dans le pire cas, les calendriers des participants ont été générés de façon à afficher une période libre alternant avec une autre occupée, et ce, de manière à ce qu'il n'y ait aucune période libre commune à tous les participants avant la période voulue selon le scénario. Ainsi, le scénario P = 1 implique 1 échange entre l'AC et chaque AP, le scénario P = 120 en implique 60, et le dernier scénario (P = 240) en implique 120.

4.3 Résultats d'expériences

Dans cette section, nous présentons les résultats des séries de tests. Chaque donnée représente une moyenne d'une dizaine de mesures effectuées sur différentes périodes de la journée durant des journées de semaine ainsi que les fins de semaine. Une variance maximale de 10% a été respectée sur un total de 400 mesures.

Nous présentons d'abord une première série de tests, ils en fournissent une analyse pour proposer ensuite une amélioration. Finalement, les résultats de toutes les sessions de mesures incluant la solution améliorée sont présentés et examinés.

4.3.1 Résultats préliminaires

Les Figures 4.6 et 4.7 reproduisent les résultats obtenus suite à l'évaluation de la charge réseau et le temps d'exécution propres aux prototypes mono-agent et multi-agents pour une planification impliquant 3 serveurs de calendriers.

Les résultats démontrent une meilleure performance de la solution multi-agents par rapport au mono-agent en terme de temps et de charge dans le cas où le succès est atteint au début de la planification. Cependant, à mesure que l'emplacement du succès s'approche de la fin de la durée de planification, cette performance est vite perdue étant donné que les mesures des deux métriques augmentent de façon linéaire pour le cas multi-agents pour excéder rapidement celles du cas mono-agent, lesquelles demeurent presque constantes.

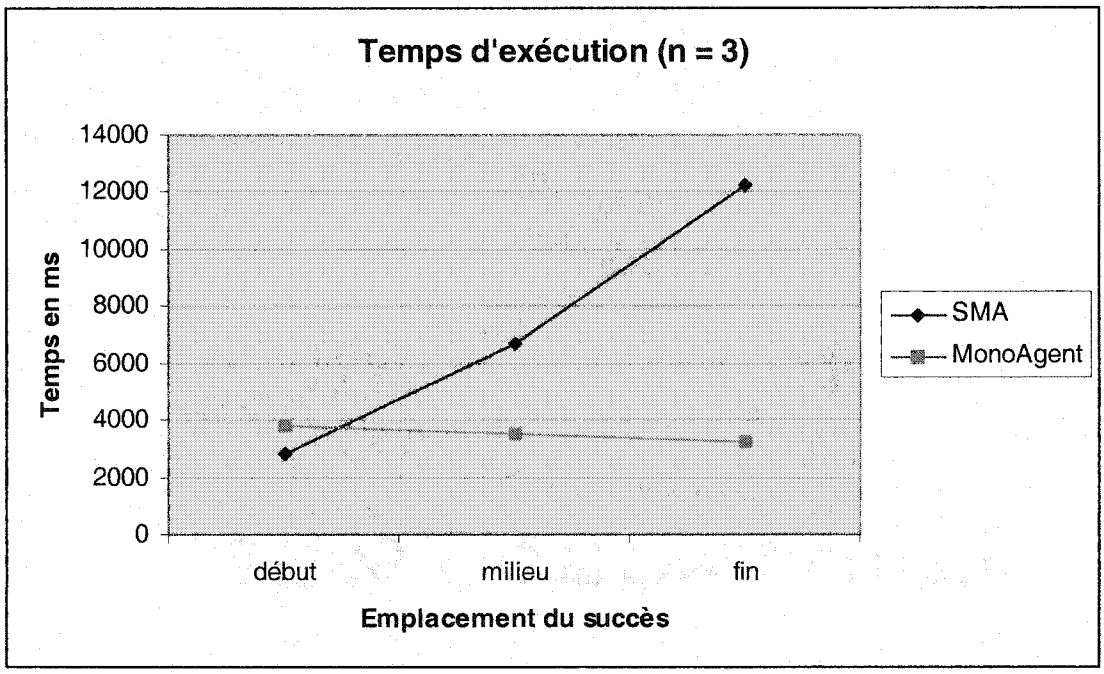

Figure 4.6 Temps de planification en fonction de l'emplacement du succès pour 3 serveurs (n=3)

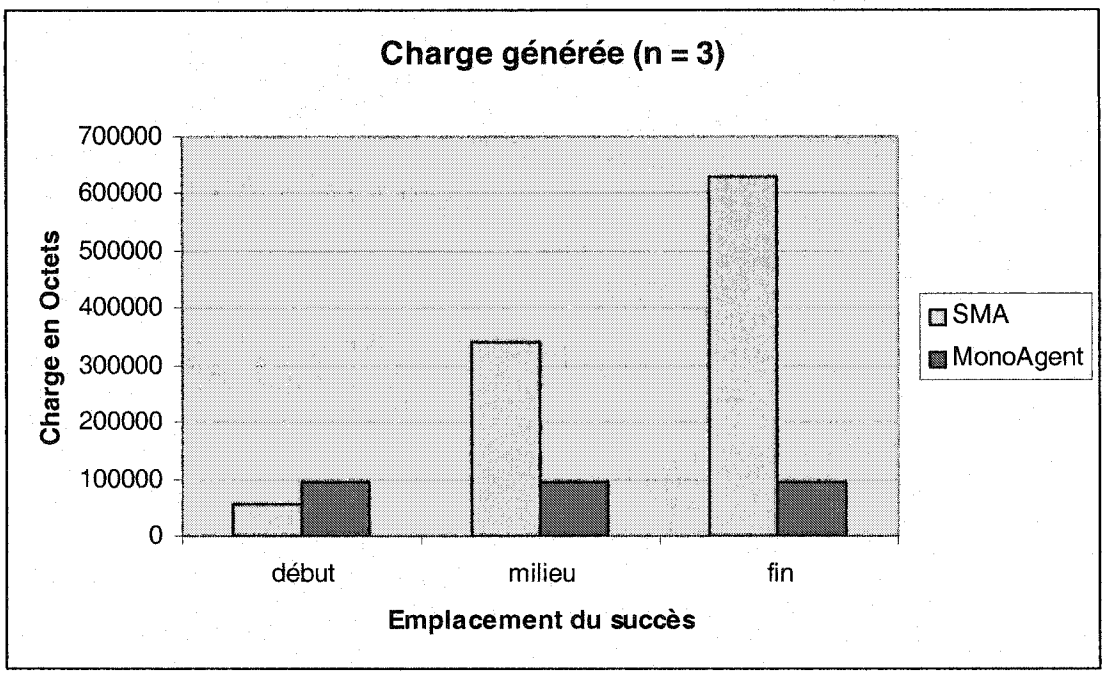

Figure 4.7 Charge générée sur le réseau en fonction de l'emplacement du succès pour 3 serveurs (n=3)

Pour expliquer la mauvaise performance de la solution multi-agents par rapport à celle mono-agent, il faut comprendre le mécanisme de communication de la plate-forme d'agents. En effet, pour protéger les interactions distantes entre les différentes agences, Grasshopper utilise les certificats X.509 et le protocole SSL (Secure Socket Layer). Ainsi, les appels RMI entre l'agent AC et les agents AP se font via le protocole SSL et toute la communication est effectuée à travers un socket SSL, chiffrée avec une clef symétrique et un algorithme de chiffrage négocié dans un échange appelé *handshake*. De plus, à chaque appel RMI, le proxy des agents AP contacte le service de domaine (Agency Domain Service) pour localiser l'agent AP, ensuite, une nouvelle connexion et une nouvelle négociation sont établies entre l'agent AC et chaque agent AP, étant donné qu'il n'existe pas de manière de constituer un canal de communication persistant entre un agent mobile et une autre entité à cause de la mobilité même de ce dernier.

En d'autres termes, l'augmentation linéaire et rapide du temps de planification et de la charge du système multi-agents est due au mécanisme de communication inter-

agent. Le poids de l'établissement d'une communication en terme de temps et de charge peut être quantifié d'après les mesures effectuées. Ainsi, nous pouvons définir le moment où le système multi-agents proposé devient moins performant que son correspondant mono-agent.

Les analyses fournies dans les paragraphes suivants découlent de l'hypothèse selon lequel le temps nécessaire à la transmission des données de planification ainsi que la charge générée par cette transmission sont, à toutes fins pratiques, négligeables. Cette hypothèse est justifiée étant donné que les données échangées sont une simple chaîne de caractères ne dépassant pas quelques octets.

Analyse en fonction du temps de réponse

La Figure 4.6 donnent les temps d'exécution de la planification multi-agents. Ces derniers sont environ de 3000 ms pour le premier scénario de test (1 échange entre l'AC et chaque AP), 7000 ms pour le deuxième (60 échanges) et 12000 ms pour le dernier (120 échanges).

Chaque mesure de temps se compose de deux parties :

- un temps constant $\boldsymbol{Tx}$ symbolisant le temps nécessaire à la migration des agents AP vers les serveurs de calendriers ;
- un temps $\boldsymbol{Ty}$ variable égale à un multiple du temps représentant le temps nécessaire à l'établissement d'une communication entre l'AC et un AP.

Ces temps sont déduits des équations suivantes :

1. $\boldsymbol{Tx} + (3 \times \boldsymbol{Ty}) = 3000$ ms.
2. $\boldsymbol{Tx} + (60 \times 3 \times \boldsymbol{Ty}) = 7000$ ms.

$\Rightarrow \boldsymbol{Ty} \approx 23$ ms.

À titre de validation, nous pouvons calculer le temps de réponse du troisième scénario selon cette estimation, ce qui nous donne un temps de :

$$\boldsymbol{Tx} + (120 \times 3 \times \boldsymbol{Ty}) \approx 11200 \text{ ms}$$

Ce résultat est proche du 12000 ms obtenu approximativement par les mesures. D'après ce résultat, la solution multi-agents devient moins performante en temps de

réponse que l'alternative mono-agent au moment où le nombre d'échanges ***Ne*** atteint une limite telle que :

$$\boldsymbol{Tx} + (\boldsymbol{Ne} \times 3 \times \boldsymbol{Ty}) > \text{T(mono-agent)} \approx 3530 \text{ ms}$$

Cela veut dire que le nombre critique d'échanges est $\boldsymbol{Ne} = 10$. Autrement dit, si la première période libre se trouve au-delà de la vingtième période (2×10), la solution multi-agents n'est plus la meilleure en terme de temps.

Analyse en fonction de la charge générée

D'après la Figure 4.7, les charges générées par la planification multi-agents lors des trois scénarios sont approximativement de 56 Ko pour le premier, 340 Ko pour le deuxième, et finalement 630 Ko pour le dernier scénario.

Similairement aux mesures de temps, chaque mesure de charge se compose de deux parties :

- une charge constante ***Cx*** symbolisant la charge produite par la migration des agents AP vers les serveurs de calendriers ;
- une charge variable égale à un multiple de la charge ***Cy*** représentant la charge créée par l'établissement d'une communication entre l'AC et un AP.

Ainsi, nous pouvons déduire la charge ***Cy*** des équations suivantes :

1. $\boldsymbol{Cx} + (3 \times \boldsymbol{Cy}) = 56$ Ko.
2. $\boldsymbol{Cx} + (60 \times 3 \times \boldsymbol{Cy}) = 350$ Ko.

$\Rightarrow \boldsymbol{Cy} \approx 1{,}7$ Ko.

Nous pouvons vérifier cette estimation en la comparant avec la mesure de la charge générée au troisième scénario. Selon cette estimation, on devrait obtenir une charge de :

$$\boldsymbol{Cx} + (120 \times 3 \times \boldsymbol{Cy}) \approx 650 \text{ Ko}$$

Ce résultat théorique est acceptable, étant donné la mesure de 630 Ko obtenue. De ce résultat, on peut déduire le nombre limite d'échanges ***Ne*** en sachant que :

$$\boldsymbol{Cx} + (\boldsymbol{Ne} \times 3 \times \boldsymbol{Cy}) = \text{C (mono-agent)} \approx 96 \text{ Ko}$$

Cela veut dire que ***Ne*** ne doit pas dépasser 10 échanges pour le cas de la charge aussi.

4.3.2 Amélioration de la solution SMA

Le modèle multi-agents présenté dans le chapitre précédent propose un scénario qui consiste à envoyer l'information renfermée dans les calendriers par parties. Cette approche émane de l'hypothèse selon laquelle la taille de l'information transmise est importante et représente une lourde charge pour le réseau. Il s'avère cependant que, pour le cas de la planification multi-parties, la faible taille en octets des informations sur les périodes d'un calendrier génère nettement moins de charge lors de sa transmission que l'établissement de la communication entre les agents.

Autrement dit, il est plus adéquat dans le cadre de cette étude d'utiliser un système multi-agents adoptant une approche qui consiste à envoyer toutes les informations utiles (les périodes libres des agendas) et ce, lors d'une seule communication. Les sous-sections suivantes fournissent les résultats de mesures justifiant le choix d'une telle approche. L'algorithme de coordination utilisé par cette dernière approche est fourni en annexe.

4.3.3 Évaluation de performance

Évaluation du temps

Les Figures 4.8 (a), 4.8 (b) et 4.8 (c) transposent les résultats de l'évaluation du temps de réponse de la tâche de planification pour les trois modèles présentés auparavant. Dans ce qui suit, nous fournissons une analyse de ces données.

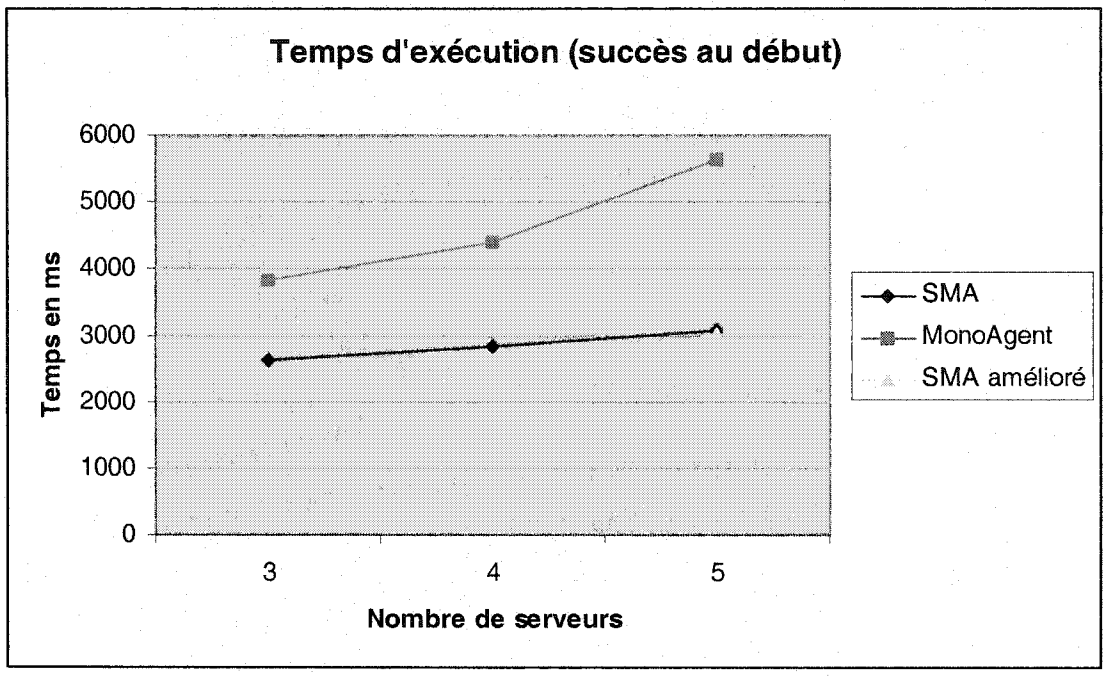

Figure 4.8 (a) Évaluation du temps d'exécution en fonction du nombre de serveurs de calendriers pour le meilleur scénario

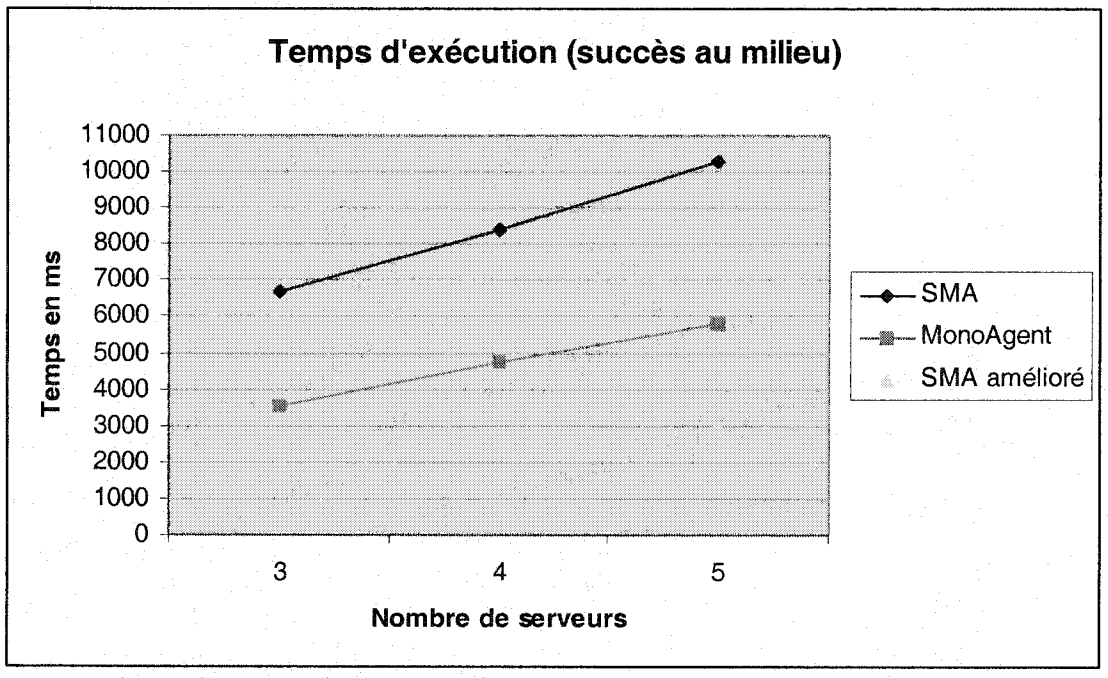

Figure 4.8 (b) Évaluation du temps d'exécution en fonction du nombre de serveurs de calendriers pour le scénario moyen

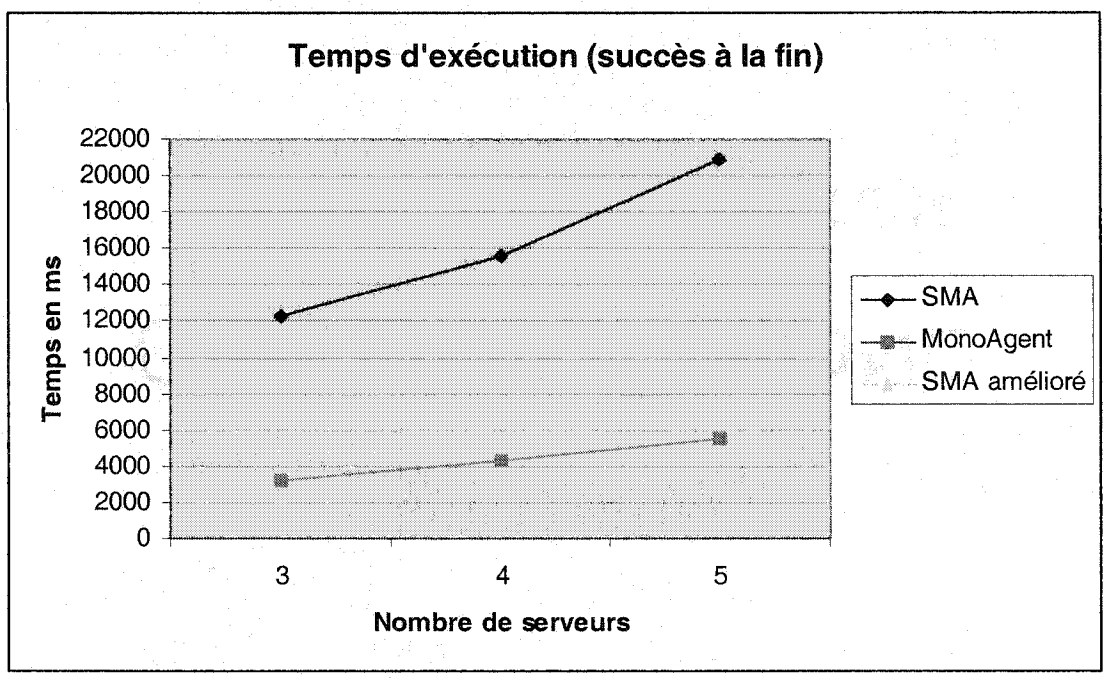

Figure 4.8 (c) Évaluation du temps d'exécution en fonction du nombre de serveurs de calendriers pour le pire scénario

L'évaluation du temps de réponse démontre que, pour les cas mono-agent et SMA amélioré, le temps de planification est pratiquement indépendant de l'emplacement du succès alors que, pour le cas du SMA, le temps est grandement affecté par ce facteur, et ce pour des raisons déjà expliquées. Les figures démontrent aussi que, pour les trois solutions, le temps de réponse augmente linéairement en fonction du nombre de serveurs de calendriers et que cette augmentation se fait avec une même pente pour les cas mono-agent et SMA amélioré, alors que la pente enregistre une augmentation pour la solution SMA. Ce dernier constat s'explique par le fait que le modèle SMA combine une dépendance vis-à-vis de l'emplacement du succès avec une autre envers le nombre de serveurs.

Le temps de réponse dans le cas du SMA amélioré est toujours le plus bas et présente une différence qui grandit légèrement par rapport à celui du cas mono-agent ; ceci est dû à la nature de la planification effectuée par ces deux modèles. En effet, si on prend le cas de trois serveurs de calendriers, la solution SMA amélioré consiste à

envoyer un agent AP (d'une taille de 10.5 KiloOctets) par serveur de calendrier, ces derniers vont à leur tour transmettre toutes les périodes libres à l'agent AC, ce qui implique donc 3 migrations et 3 transmissions. Pour le cas mono-agent, l'agent *MonoAgentPlanificateur* (d'une taille de 11.8 KiloOctets) suit un itinéraire qui consiste à visiter chacun des trois serveurs pour retourner vers la machine principale tout en transportant l'information contenant les périodes libres, ce qui implique en gros 4 migrations et 4 transmissions. Pour résumer, le temps SMA amélioré est toujours plus petit que celui du mono-agent parce qu'il implique une migration et une transmission de données en moins, et cette différence va en grandissant vu que la taille des agents migrants est plus petite pour le premier cas que pour le second.

Évaluation de la charge

Les Figures 4.9 (a), 4.9 (b) et 4.9 (c) illustrent les résultats de mesures de charges effectués.

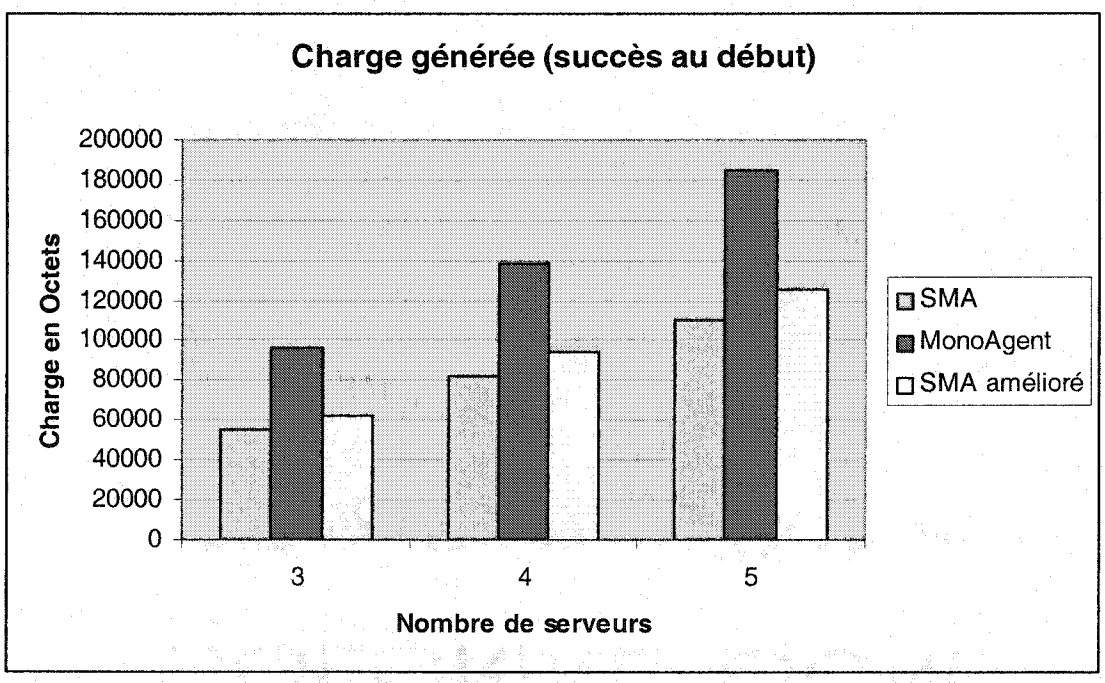

Figure 4.9 (a) Évaluation de la charge générée en fonction du nombre de serveurs de calendriers pour le meilleur scénario

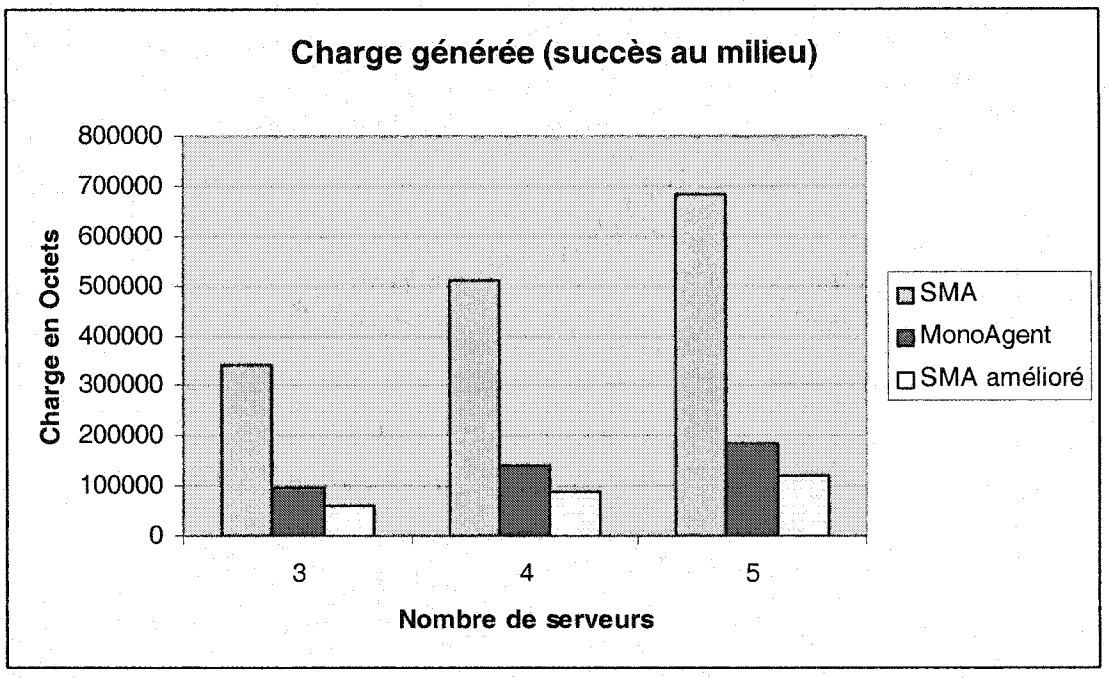

Figure 4.9 (b) Évaluation de la charge générée en fonction du nombre de serveurs de calendriers pour le scénario moyen

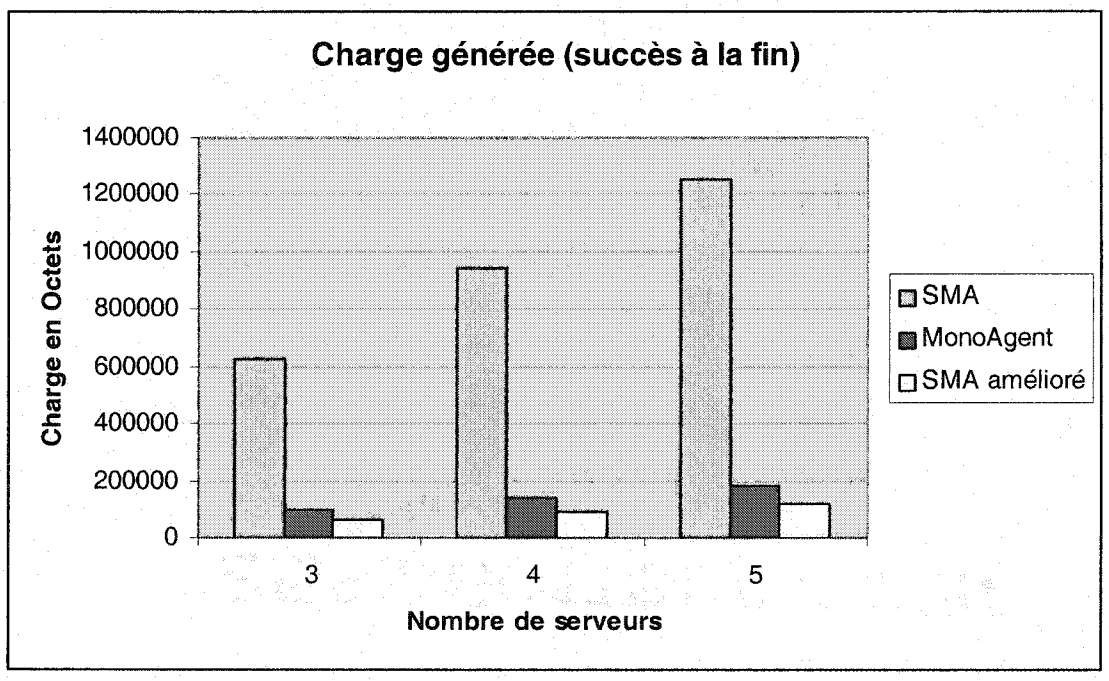

Figure 4.9 (c) Évaluation de la charge générée en fonction du nombre de serveurs de calendriers pour le pire scénario

Pour ce qui est de l'évaluation de la charge générée par les trois différents modèles, nous pouvons remarquer les mêmes tendances constatées lors de l'évaluation du temps de réponse, mis à part une seule exception. En effet, pour le premier et meilleur scénario, c'est la solution SMA et non la solution SMA amélioré qui génère la charge la moins importante, ce qui n'est pas le cas pour la mesure du temps. Ce constat peut être expliqué par les faits suivants :

1. la différence de taille entre les agents AP des deux modèles (10.5 KiloOctets pour le SMA amélioré et 11.1 KiloOctets pour le SMA) fait que la migration des agents pour le premier requiert moins de temps que pour le second ;
2. la solution SMA amélioré consiste à envoyer toutes les périodes libres, alors que celle du SMA n'en envoie qu'une seule (ce qui implique une information de taille 120 fois plus grande pour le premier cas), la première solution génère donc plus de charge.

La différence de taille entre les agents AP de la solution SMA et ceux de la solution SMA amélioré s'explique par le fait que ces derniers possèdent un module de planification et d'accès au calendrier plus simplifié, étant donné que ces derniers n'ont besoin que d'identifier toutes les périodes libres afin de les envoyer. Dans le cas des agents AP de la solution SMA, ces derniers doivent repérer à chaque fois la première période libre en partant de celle communiquée par l'agent AC. En d'autres termes, ils possèdent plus d'intelligence, ce qui nécessite une plus grande taille.

4.3.4 Évaluation de la charge et du temps pour des données de grande taille

Le modèle SMA proposé au départ repose, comme précisé précédemment, sur l'hypothèse que l'information transmise est de taille importante. Pour vérifier la validité de ce propos, il faut évaluer les performances des trois modèles sous une telle condition.

Les Figures 4.10 (a) et 4.10 (b) illustrent les mesures de temps et de charge obtenues pour une planification impliquant des données transmises d'une taille 1000 fois supérieure à celle impliquées dans les mesures précédentes.

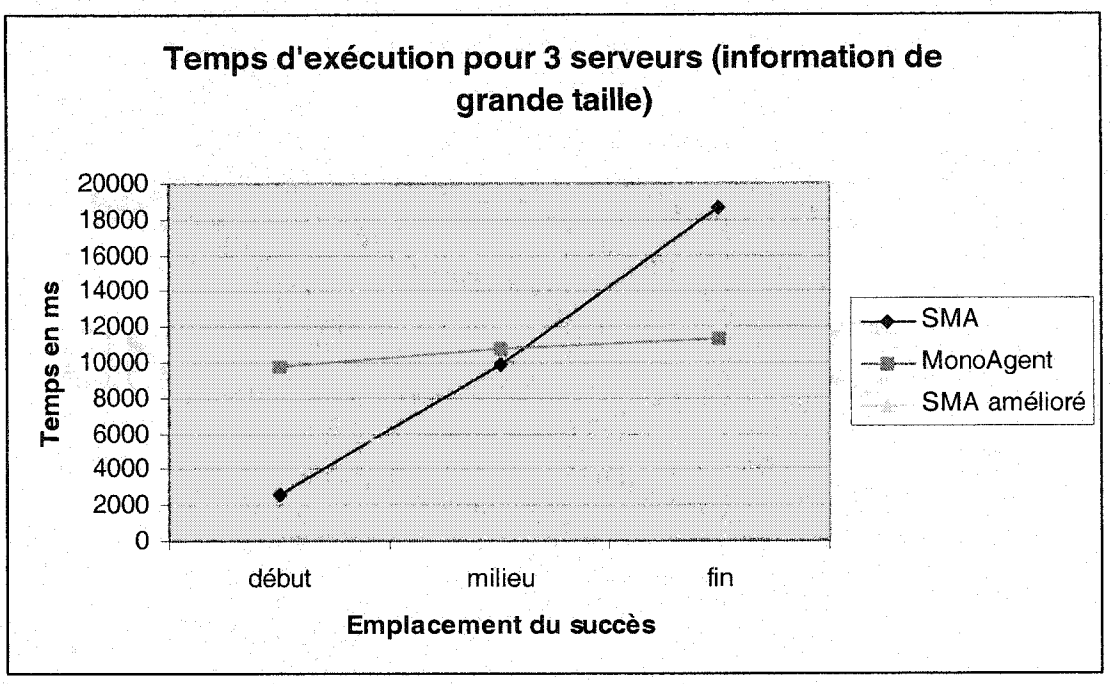

Figure 4.10 (a) Temps de planification en fonction de l'emplacement du succès pour des données de grande taille et 3 serveurs (n=3)

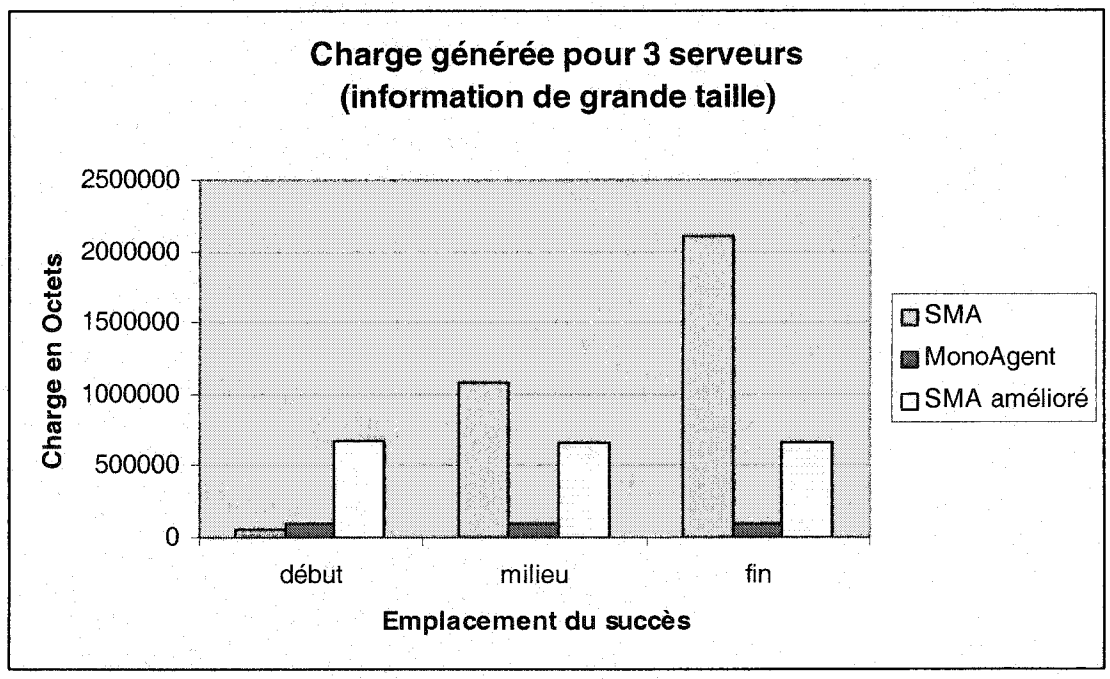

Figure 4.10 (b) Charge générée en fonction de l'emplacement du succès pour des données de grande taille et 3 serveurs (n=3)

Le résultat de mesures de temps de réponse démontre que la performance du modèle SMA proposé au départ s'améliore par rapport à celle des deux autres modèles. En effet, le temps de planification de la solution SMA est meilleur que ceux du SMA amélioré et du mono-agent pour le premier scénario, il reste plus bas que celui du mono-agent pour le deuxième scénario, pour finalement devenir le plus important lors du pire cas. Cette amélioration est due à la combinaison de deux facteurs : le premier est l'accroissement du temps nécessaire pour transmettre toute l'information dans le cas du SMA amélioré et celui de transporter cette information pour le cas mono-agent. Le deuxième facteur est la baisse relative (par rapport à la taille des données transmises) du temps nécessaire à l'envoi de messages inter-agent, ce phénomène est dû au fait que la communication ne souffre plus de la latence de l'implémentation TCP, étant donné que la taille des paquets transmis dépasse dans ce cas celle du segment TCP sur Ethernet qui est de 1472 Octets. En effet, dans le cas de données de faible taille, à cause de l'implémentation TCP, il faut attendre que la taille du paquet à transmettre dépasse celle de la fenêtre ou qu'un autre paquet destiné au même nœud transite par la machine. Au cas où aucun de ces évènements ne survient, la machine transmet le paquet ou la confirmation de réception après 200 ms, ce qui implique une latence de 400 ms pour un aller-retour.

Pour ce qui est des mesures de charge, la Figure 4.10 (b) démontre qu'en comparaison avec le cas où les données échangées étaient de faible taille, la performance du modèle SMA ne s'est pas améliorée par rapport à celle des autres. Ceci est dû au fait que le coût en octets de l'établissement des multiples communications reste toujours élevé. Ceci dit, cette dernière mesure de charge révèle une différence par rapport aux précédentes. En effet, la solution SMA amélioré génère dans ce cas une charge plus élevée que celle du mono-agent. Ce dernier constat s'explique par le fait que la plate-forme d'agents Grasshopper exécute un algorithme de compression sur le code de l'agent et son contenu avant de le transférer d'une agence à une autre. Ainsi, la taille des données transportées par l'agent dans le cas du modèle mono-agent se trouve réduite grâce à la compression. Ceci n'est pas le cas pour les données transmises par les agents

AP du modèle SMA amélioré qui sont envoyées comme réponse à un appel RMI et ne bénéficient d'aucune compression. Cette constatation n'a pu être faite lors des expériences impliquant des données de faible taille, étant donné que la compression de ces dernières ne générait aucune différence sur les mesures.

4.4 Synthèse des résultats

Les prototypes mono-agent et multi-agents que nous avons réalisés ont permis la comparaison de performances d'approches multi-agents par rapport à celle de mono-agent dans le cadre d'applications réparties. Cette évaluation de performance s'est basée sur la planification d'événements multi-parties comme application et a été effectuée en se basant sur deux facteurs : la charge totale générée sur le réseau et le temps requis pour l'accomplissement de la tâche de planification.

Les mesures effectuées ont permis de constater l'inconvénient que représentait le fait de ne pas pouvoir établir de communication persistante entre les agents d'un SMA et le coût inhérent en termes de temps et de charge à l'établissement de multiples connexions lors d'échanges de messages inter-agent, empêchant ainsi de profiter pleinement du fractionnement des données nécessaires à l'accomplissement d'une tâche répartie que permettent les SMA. Cependant, nous avons pu constater que cet inconvénient peut être contourné en minimisant le nombre d'échanges permettant ainsi au SMA de démontrer un gain de performance distinctif par rapport au mono-agent et ce, pour tous les scénarios. Ce gain de performance enregistre néanmoins un recul en terme de charge générée lorsque les données échangées sont de grande taille. Toutefois, ce problème peut être écarté en utilisant la compression sur les données envoyées lors d'échanges inter-agent.

Finalement, les systèmes multi-agents présentent une plus grande robustesse par rapport à leur équivalent mono-agent, car ces derniers permettent de garder un historique de l'exécution de la tâche et peuvent la reprendre en cas de bris de communication ou de perte d'un ou de plusieurs agents, ce que ne permet pas bien sûr un système mono-agent.

CHAPITRE V

CONCLUSION

Plusieurs applications réseau nécessitent un accès à des informations réparties sur plusieurs serveurs de données. Dans ce contexte, le paradigme agent mobile peut représenter un choix intéressant. Dans ce mémoire, nous avons évalué les possibilités qu'offrait ce paradigme en comparant des modèles mono-agent et multi-agents. La planification d'événements a été retenue comme application répartie, et les mesures de temps et de charge ont servi de facteurs à l'appréciation des performances. Dans ce chapitre, nous présentons les principaux apports des travaux effectués dans le cadre de ce mémoire. Nous fournissons ensuite une synthèse des problèmes ouverts, pour terminer par des indications de travaux futurs et les perspectives intéressantes pour nos travaux.

5.1 Contributions principales

Dans le but d'évaluer les possibilités qu'offrait le paradigme agent mobile, nous avons proposé d'appliquer des solutions mono-agent et multi-agents pour résoudre la tâche de planification d'événements électroniques. Cette dernière est un bon exemple des applications réparties, elle nécessite l'accès à des agendas enregistrés sur différents serveurs représentant des bases de données résidant dans des domaines différents. Des prototypes reproduisant chaque solution ont été élaborés pour les fins de l'évaluation.

La conception d'une architecture multi-agents utilisant un algorithme de coordination inter-agent et employant une partie d'une ontologie élaborée pour l'accomplissement de la planification a permis de constater la capacité des SMA à s'adapter à des tâches réparties; elle démontre aussi, grâce au partage centralisé de l'information, la robustesse et le degré d'automatisation qu'offrent de tels systèmes.

Les mesures de temps nécessaire à l'exécution de l'application répartie et les mesures de charge générée sur le réseau par cette dernière ont permis de préciser le coût des échanges entre agents dans un système multi-agents et de déterminer l'utilité du recours à la partition des données traitées en fonction de la taille de ces données et du degré de granularité choisi. Ainsi, nous avons pu constater avec le SMA un gain de performance pour les deux métriques par rapport au mono-agent en réduisant le nombre de communications. Toutefois, ce constat change en augmentant la taille des données traitées étant donné que le SMA perd de sa performance en terme de charge, et ce, à cause du phénomène de compression pratiquée par la plate-forme sur le code des agents. Cependant, nous estimons que cette perte peut être corrigée en appliquant une compression des données échangées entre les agents du SMA.

5.2 Synthèse des problèmes ouverts

Le principal problème rencontré par les systèmes SMA et mono-agent pour la planification d'événements multi-parties est celui de la sécurité. Les agents planificateurs ayant accès aux agendas des participants peuvent effectuer des transactions malicieuses tels que l'altération des informations des agendas ou leur dévoilement à de tierces parties, que ce soit de façon délibérée ou non. En effet, le fait que les serveurs de calendriers doivent reposer sur une plate-forme pour agents rend ces derniers, ainsi que les informations qu'ils contiennent, vulnérables face à certains types d'attaques.

Les prototypes SMA et mono-agent exigent que les serveurs de calendriers soit munis de la plate-forme Grasshopper, ce qui représente une limitation pour ces derniers. En effet, en plus de diminuer la portabilité de l'application basée sur des architectures à base d'agents, la plate-forme utilise certains ports de communications (7000-7002), ce qui représente une faille supplémentaire de sécurité. Un autre problème lié à l'utilisation d'une telle plate-forme est que les agents planificateurs doivent être codés dans le même langage que cette dernière, c'est-à-dire Java. Ce langage n'est pas nécessairement celui des APIs de logiciels de calendriers (par exemple, celui de MS Outlook est en VB).

Une autre limitation aux solutions proposées est qu'elles ne considèrent pas les préférences des participants et des facteurs tels que des réunions déplaçables ou des participants optionnels. Ces solutions réduisent en effet le problème de planification et s'appliquent uniquement à trouver la première période libre commune à tous les usagers.

5.3 Travaux futurs

L'évaluation de performance des deux prototypes à base d'agents n'a considéré qu'un seul format d'agenda (le format XML). Il serait cependant intéressant de considérer le cas où plusieurs formats d'agendas électroniques sont utilisés par des serveurs. Nous pouvons déjà anticiper l'avantage que présentera dans ce cas là les SMA sur les modèles mono-agent. En effet, tandis que pour le cas mono-agent, l'agent de planification doit implémenter plusieurs librairies dépendamment des APIs des calendriers existants sur les serveurs, la solution SMA permet de recourir à des agents planificateurs « spécialisés », c'est à dire que chacun d'entre eux implémente uniquement la librairie nécessaire à l'accès au format d'agendas utilisé par le serveur qui lui est associé. Ainsi, la taille des agents du SMA serait moins importante que celle du modèle mono-agent.

Les solutions proposées ne considèrent pas non plus les scénarios de conflit entre plusieurs systèmes de planification. En effet, nous pouvons imaginer le cas où deux systèmes ou plus planifient un événement impliquant un ou plusieurs calendriers en commun. Dans ce cas, on pourrait se retrouver devant la situation où un système réserve une période pour un événement alors que les autres systèmes la perçoivent comme étant toujours libre. En guise de travaux futurs, il serait donc essentiel de considérer l'interaction entre différents agents appartenant à plusieurs systèmes de planification qui s'exécutent en parallèle.

Aussi, étant donné que les agents AP ne migrent plus une fois rendus au serveur, il est théoriquement possible d'établir une communication de type persistante entre les agents, sans pour autant imposer aux parties d'être connectées durant toute l'interaction, car cette dernière caractéristique représente un des grands avantages du paradigme agent

mobile. En combinant ce fait avec une compression de données, on réussirait ainsi à diminuer le coût de la communication et mieux profiter de la granularité que permettent les applications basées sur des SMA.

Une autre perspective intéressante serait de considérer l'accès nomade par un utilisateur mobile, ce cas se présente quand l'organisateur lance la planification par exemple à partir d'un terminal PDA (*Personal Data Assistant*) et que ce dernier change de point d'accès. C'est en effet une visée utile étant donné l'avenir que promet le service nomade au paradigme agent mobile.

Par ailleurs, la planification d'événements n'est qu'un exemple d'application répartie. Une suite logique à nos travaux serait d'étendre l'étude à d'autres applications de ce type telles la recherche d'informations ou le commerce mobile. Ceci permettrait de vérifier si les résultats sur les performances des SMA et mono-agent peuvent être généralisés à tout le domaine de l'accès à l'information répartie.

BIBLIOGRAPHIE

Ashir A., Joo K. H., Kinoshita T. et Shiratori N., ''Multi-agent based decision mechanism for distributed meeting scheduling system''. In *Proceedings of the 1997 International Conference on Parallel and Distributed Systems (ICPADS '97),* Séoul, décembre 1997.

Bond A. H. et Gasser L., editors. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers : San Mateo, CA, 1988.

Bourron T., ''Application des systèmes multi-agents dans les télécommunications : États de l'art, enjeux et perspectives''. paru dans *Principes et architecture des Systèmes multi-agents*, édité par Briot et Demazeau, Chez Hermes, 2001.

Ciancarini P., Omicini A. et Zambonelli F., ''Coordination Models for Multi-Agent Systems''. In AgentLIink News 3, juillet 1999. pp.3-6. www.agentlink.org.

Côté M., ''Une architecture multi-agent et son application aux systèmes financiers''. Master's thesis, Département d'Informatique, Université Laval, Janvier 1999.

Ferber J. et Ghallab M., ''Problématiques des univers multi-agents Intelligents''. In *Actes des journées nationales PRC-GRECO Intelligence Artificielle*, pages 295-320, Toulouse, mars 1988.

Ferber J., *Conception et Programmation par Objets.* Technologies de pointe – informatique, Hermes, 1990.

Finin T. et Fritzson R., ''KQML: a language and protocol for knowledge and information exchange''. In Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence, pp. 126-136, Lake Quinalt, WA, juillet 1994.

Franklin S. et Graesser A., ''Is it an agent, or just a program ? : A taxonomy for autonomous agents''. In J. P. Mueller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III : Theories, Architectures, and Languages (LNAI Volume 1193),* pages 21-35. Springer-Verlag : Heidelberg, Germany, 1997.

Garrido L. et Sycara K., ''Multi-Agent Meeting Scheduling : Preliminary Experimental Results''. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS '96),* 1996, pages 95 - 102.

Glitho R.,Olougouna E. et Pierre S., ''A Mobile-Agent and Their Use for Information Retrieval : A Brief Overview and an Elaborate Case Study''. IEEE Network, January/February 2002, pp.34 - 41.

Gruber T. R., ''A translation approach to portable ontologies''. *Knowledge Acquisition,* 5(2). pp. 199-220, 1993. http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html .

Harisson C. G., Chess D. et Kerskenbaum A., ''Mobile Agents : Are they a good idea ?''. IBM Research Report, RC19887, mars 1995.

Huhns M. N. et Stephens L. M., ''Multiagent Systems and Societies of Agents''. A Modern Approach to Distributed Artificial Intelligence''. G. Weiss, MIT Press, 1999.

Iglesias C.A., Garijo M., Gonzàlez J.C., et Velasco J.R., ''Analysis ans design of multi-agent systems using mascommonkads''. In *AAAI'97 Workshop on Agent Theories, Architectures and Languages*, Providence, RI, 1997.

Jeong W-S., Yun J-S. et Jo G-S., ''Cooperation in Multi-Agent System for Meeting Scheduling''. In *Proceedings of TENCON'99*, pages 832-835, CheJu, korea, septembre 1999.

Kotz D. et Gray R., ''Mobile Agents and the Future of Internet''. In ACM Operaqting System Review, pp. 7-13, août 1999.

Labidi S. et Lejouad W., ''De l'Intelligence Artificielle Distribuée aux Systèmes Multi-Agents''. Projet SECOIA, Rapport de recherche n° 2004, août 1993.

Olougouna E., Architectures évolutives pour applications d'agendas électroniques basées sur les agents mobiles. École polytechnique de Montréal, juin 2001.

Roth B. H., Hewett M., Washington R., Hewett R., et Seiver A.. ''Distributing intelligence within an individual''. In *Distributed Artificial Intelligence*, volume 2 of Research Notes in Artificial Intelligence, pp. 385-412. Pitman, 1989.

Troudi N., ''Un système multi-agent pour les environnements riches en information''. Master's thesis, Département d'Informatique, Université Laval, Avril 1999.

Weiss G., ''Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence''. G. Weiss, MIT Press, 1999.

Zeng D. et Sycara K., ''Coordination of multiple intelligent software agents''. *International Journal of Cooperative Information Systems*, 5 . pp.181-212, 1996.

Sites Web

Agentland	http://www.agentland.com
AgentLink	www.agentlink.org
Aglet	www.ibm.com/aglet
Grasshopper	www.grasshopper.de
GUARDIAN	www-ksl.stanford.edu/projects/guardian/index.html
Java	www.java.sun.com
Lotus notes	www.lotusnotes.com
Lotus XML Toolkit	www.lotus.com/developers/devbase.nsf/homedata/homexmltk
Microsoft Outloook	www.microsoft.com/office/outlook
MSN calendar	calendar.fr.msn.ca
Ontologies	agents.umbc.edu/aw/Topics/Communicative_Agents/Ontologies /index.shtml
Voyager	www.objectspaces.com
SMA	www.multiagent.com

ANNEXES

Algorithme de coordination utilisé par la solution SMA amélioré :

Initialisation :

adresses usagers : $(U_1,\ldots,U_k)$.
serveurs : $(S_1:\{U_1,\ldots,U_r\} ,\ldots\ldots, S_i:\{U_s,\ldots,U_t\} , S_n:\{U_{t+1},\ldots,U_k\})$.
période : $P = [J_{début} , J_{fin}]$.
durée : D.
nombre de périodes : $m = P/D = (J_{fin} - J_{début})/D$
tableau de périodes libres : PDL[m] = NULL .
réussite : = FAUX.
temps maximal d'exécution : = T_{max} .

Début

POUR i : = 1 **À** n
Créer agent planificateur AP_i (D, PDL[m], P)
Envoyer AP_i vers S_i
Réclamer PDL_i [ ] de AP_i
Fin **POUR**
j : = 1
TANT QUE (réussite = FAUX) **ET** ($j \leq m$) **ET** ($t \leq T_{max}$)
trouver_date : = 1
i : = 1
TANT QUE ($i \leq n$)
SI AP_i a retourné PDL_i [ ] **ALORS**
SI ($i \neq 1$) **ALORS**
SI ($PDL_i [j] = PDL_{i-1}[j]$)
trouver_date : = trouver_date + 1
Fin **SI**
i : = i + 1
SINON AP_i a été perdu
Créer un nouveau AP_i (D, PDL[ ], P) et l'envoyer à S_i
Réclamer PDL_i [ ] de AP_i
Fin **SI**
Fin **TANT QUE**
SI (trouver_date = n) **ALORS**
Réussite : = VRAI
PDL : = PDL_i [j]
SINON
j = j + 1
Fin **TANT QUE**
SI (réussite = VRAI) **ALORS**
POUR i : = 1 **À** n
Réclamer à AP_i de notifier tout $U \in S_i$ d'une réunion à PDL
Ordonner à AP_i de se supprimer
Fin **POUR**
SINON (échec de communication ou de consensus sur une date)
Notifier l'organisateur U_i de la nature de l'échec.
Fin **SI**
FIN